
Modeling TCP Performance

Lecturer: Roman Dunaytsev
dunaytse@cs.tut.fi

Outline

- ⌘ Background
- ⌘ Related work
- ⌘ Modeling connection establishment phase
 - ⊞ Model by N. Cardwell et al.
- ⌘ Modeling the initial slow start phase
 - ⊞ Model by N. Cardwell et al.
 - ⊞ Model by D. Zheng et al.
 - ⊞ Model by B. Sikdar et al.
 - ⊞ Proposed model
- ⌘ Modeling TCP steady state throughput
 - ⊞ Model by M. Mathis et al. ("square-root" formula)
 - ⊞ Model by J. Padhye et al. (PFTK-model)
 - ⊞ PFTK-model revised
- ⌘ Modeling TCP NewReno steady state throughput
 - ⊞ TCP NewReno variants: Slow-but-Steady & Impatient
 - ⊞ Proposed model
- ⌘ TCP fairness under loss synchronization
- ⌘ Network asymmetry & TCP performance

Background

TCP function	What is used
Data transfer between application processes	IP service
Multiplexing/demultiplexing	Port numbers
Reliability	Checksum computation Connection establishment and termination Sequence numbers Acknowledgements Protect against wrapped sequence numbers Maximum segment lifetime Retransmissions RTT and RTO computation, Karn's algorithm
Flow control	Receive window Sliding window Silly window syndrome avoidance Nagle's algorithm
Congestion control	Slow start Congestion avoidance Fast retransmit Fast recovery ECN-support

Background (cont'd)

- ⌘ TCP is defined in a number of RFCs
- ⌘ **RFC 4614 “A Roadmap for Transmission Control Protocol (TCP) Specification Documents”**
 - ☑ It provides a brief summary of the RFC documents that relate to TCP
- ⌘ Recall that not every RFC is a standard
- ⌘ The Status field gives the RFC’s current status:
 - ☑ Standards track (proposed standard, draft standard, or Internet standard)
 - ☑ Experimental
 - ☑ Informational
 - ☑ Best current practice (BCP)

Background (cont'd)

⌘ Why TCP?

⌘ TCP is the prevalent transport layer protocol on the Internet

- ☑ Most Internet applications, such as WWW, email, FTP, and P2P file sharing, use TCP to provide reliable data transfer over unreliable "best-effort" service of IP
- ☑ Since these applications are the dominant applications in today's Internet, as a result, TCP controls a large fraction of bytes and packets traversed over the Internet

⌘ About 90% of all packets and bytes are TCP traffic

☑ E.g., <http://netflow.internet2.edu/weekly/>

☑ February 25, 2008:

Protocols	Octets	Packets
ICMP[1]	0.05% 393.2G	0.27% 2.999G
IGMP[2]	0.00% 40.60M	0.00% 1.196M
IP-ENCAP[4]	0.00% 22.55G	0.00% 27.77M
TCP[6]	90.27% 724.8T	86.03% 967.2G
UDP[17]	6.43% 51.64T	11.18% 125.7G
IPv6[41]	0.00% 19.80G	0.01% 78.29M
GRE[47]	2.55% 20.46T	1.90% 21.32G
ESP[50]	0.68% 5.435T	0.60% 6.693G
AX.25[93]	0.00% 178.2k	0.00% 1.600k
PIM[103]	0.00% 4.071G	0.00% 40.23M
IPMP[169]	0.00% 4.161M	0.00% 57.80k
Other	0.01% 110.3G	0.02% 205.6M
Total	100.00% 802.9T	100.00% 1.124T

Background (cont'd)

⌘ However, the total number of UDP flows is approximately the same as the total number of TCP flows

⌘ E.g., www.slac.stanford.edu/comp/net/slac-netflow/html/netflow.html

⌘ December 24, 2007:

Protocol	Total Recs	Total Flows	Total Pkts	Total Bytes
ALL	7.07 M	9.15 M	3.58 G	3.58 T
TCP	4.43 M 62.62 %	5.40 M 59.01 %	3.55 G 99.11 %	3.57 T 99.62 %
UDP	2.39 M 33.81 %	3.34 M 36.54 %	17.05 M 0.48 %	7.22 G 0.20 %
GRE	11.78 K 0.17 %	32.75 K 0.36 %	12.81 M 0.36 %	5.68 G 0.16 %
ICMP	240.84 K 3.41 %	0.37 M 4.09 %	2.11 M 0.06 %	766.21 M 0.02 %

⌘ Or can be up to 2 times greater than flow count of TCP traffic

⌘ *Myung-Sup Kim, Young J. Won, and James W. Hong. "Characteristic analysis of internet traffic from the perspective of flows". Computer Communications, volume 29, issue 10, June 2006, pp. 1639-1652*

⌘ The number of UDP flows with a couple of packets takes about 90% of the total UDP flows

⌘ Most of them are generated by P2P applications and some abnormal traffic such as DoS attacks and Internet worms

Background (cont'd)

Studying protocol performance	Advantages	Disadvantages
Measurements in real-life environment	Capture real protocol behavior The most realistic results are given by live Internet tests	Require access to the network equipment Privacy issues arise Noncontrollable environment Large diversity of protocols/applications/user behavior patterns complicates the analysis No replication is possible
Modeling in network simulation environment	Ease of use and flexibility Configurable parameters Different levels of abstractions Fast results Replication	Unknown relationship to Internet reality No commonly accepted methodology exist Simulation parameter settings affect results Implementation bugs
Analytical modeling	Ease of use and flexibility Configurable inputs Different levels of abstractions Instant results	Unknown relationship to Internet reality No commonly accepted methodology exist Modeling assumptions affect results Errors

Background (cont'd)

⌘ **Modeling and simulations must go hand-in-hand with measurement**

- ⊞ *S. Floyd, E. Kohler. "Internet research needs better models". ACM SIGCOMM Computer Communication Review, volume 33, issue 1, January 2003, pp. 29-34*

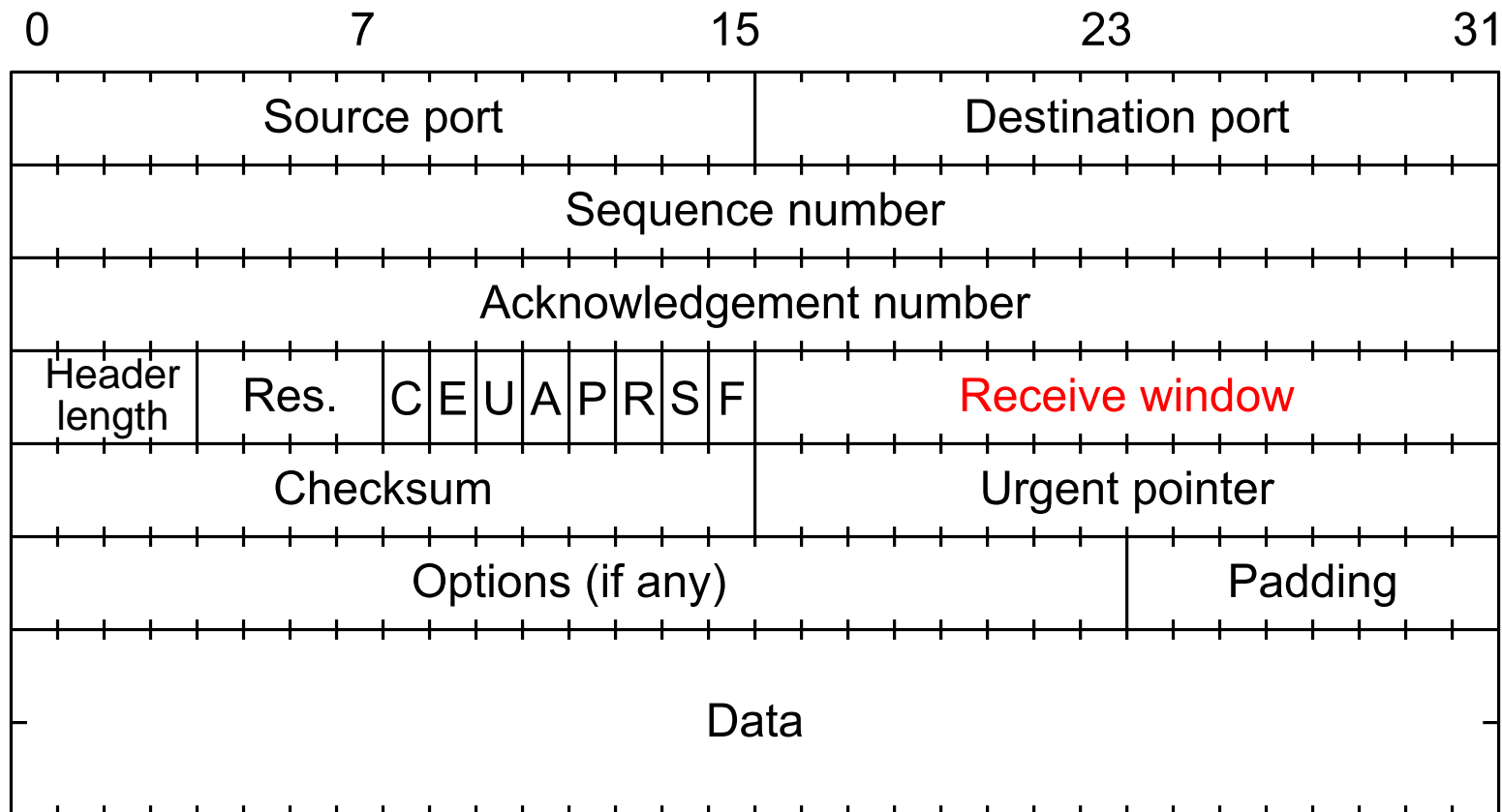
⌘ **Three instances of TCP:**

- ⊞ Specified in RFCs and technical documentation
 - ⊞ Implemented in real operating systems
 - ⊞ Implemented in network simulators
- ⌘ Each implementation introduces its own interpretation of the standards, its own choice of unspecified parameters, and, sometimes, its own bugs
- ⊞ All of these factors create differences among the instances of TCP
- ⌘ As a rule, analytical models assume the TCP implementation complies with the corresponding RFCs
- ⌘ A number of specific issues related to differing TCP implementations arise:
- ⊞ Development of accurate analytical models and
 - ⊞ Accuracy of network simulators
 - ⊞ Validation of proposed models
- ⌘ Let us consider this problem with respect to the TCP receive window parameter

Background (cont'd)

⌘ Receive window (rwnd), 16 bits

- ☑ Specifies the number of bytes the sender of the segment is ready to accept
- ☑ The maximum value is $2^{16} - 1 = 65,535$ bytes
- ☑ The sender must not transmit more data than it was defined by the receiver
- ☑ If there is no free room in the receive buffer, the receiver sends ACK with $rwnd = 0$



Background (cont'd)

- ⌘ http://www.w3schools.com/browsers/browsers_os.asp
- ⌘ Microsoft Windows XP is the most popular operating system

OS Statistics - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Refresh Home

Address http://www.w3schools.com/browsers/browsers_os.asp

Web Quality

W3Schools Forum

Helping W3Schools

Ads by Google

- [Operating System](#)
- [OS](#)
- [W3SCHOOLS.com](#)
- [SuSE Linux](#)
- [Computer Mouse](#)

OS Platform Statistics

Windows XP is the most popular operating system. The windows family counts for nearly 90%:

2008	WinXP	W2000	Win98	Vista	W2003	Linux	Mac
February	72.3%	4.0%	1.0%	7.6%	1.8%	3.8%	4.3%
January	73.6%	4.0%	0.8%	7.3%	1.9%	3.6%	4.4%

2007	WinXP	W2000	Win98	Vista	W2003	Linux	Mac
November	73.8%	5.1%	1.0%	6.3%	2.0%	3.3%	3.9%
September	74.3%	5.4%	0.9%	4.5%	2.0%	3.4%	3.9%
July	74.6%	6.0%	0.9%	3.6%	2.0%	3.4%	4.0%
May	75.0%	6.5%	0.9%	2.8%	1.9%	3.4%	3.9%
March	76.0%	7.2%	0.9%	1.9%	1.9%	3.4%	3.8%
January	76.1%	7.7%	1.0%	0.6%	1.9%	3.6%	3.8%

Internet

Background (cont'd)

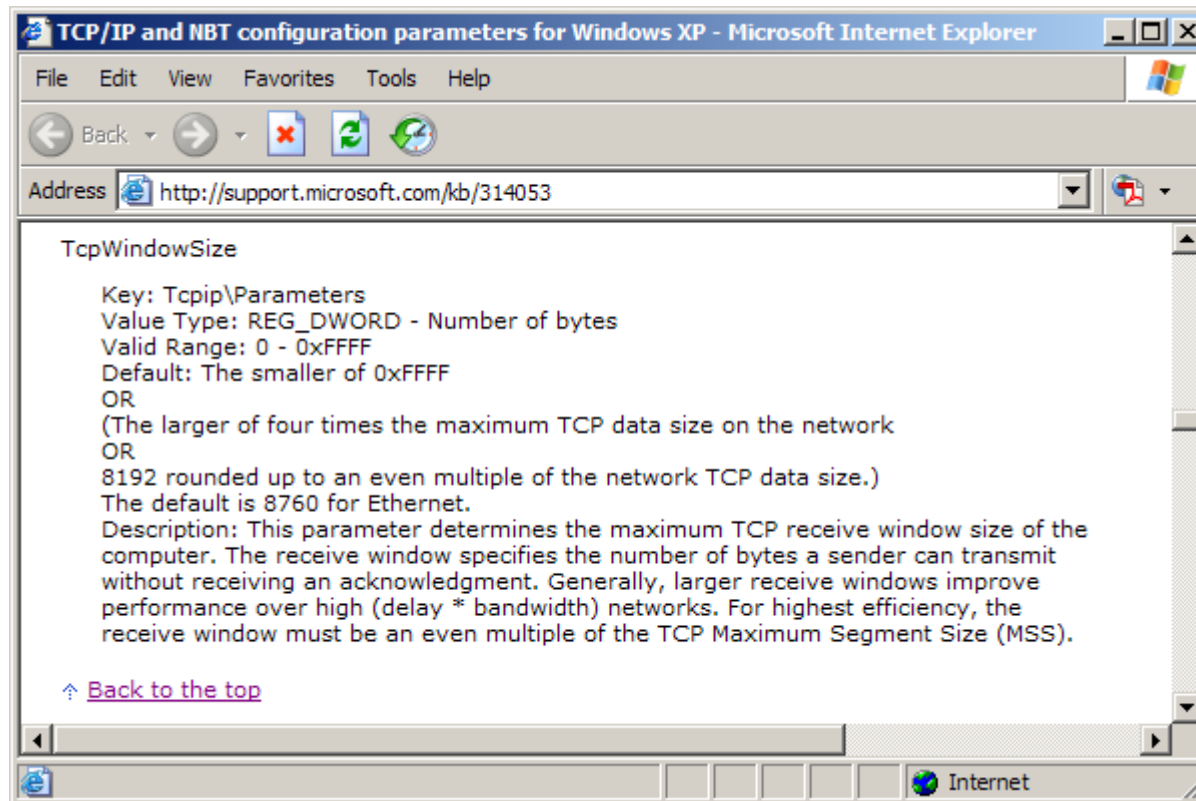
⌘ <http://support.microsoft.com/kb/314053>

⌘ “TCP/IP and NBT configuration parameters for Windows XP”

☑ Article ID: 314053

☑ Last review: November 17, 2006

☑ Applies to: Windows XP Home Edition, Windows XP Professional



Background (cont'd)

- ⌘ A TUT computer (Microsoft Windows XP Professional, 512 MB of RAM)

The screenshot shows the Wireshark interface with a capture of two packets. Packet 2 is selected, showing its details in the packet list and packet bytes panes.

No.	Time	Source	Destination	Protocol	Info
1	0.000000	130.230.52.139	217.70.129.242	TCP	optima-vnet > http
2	0.039661	217.70.129.242	130.230.52.139	TCP	http > optima-vnet

Packet 2 details:

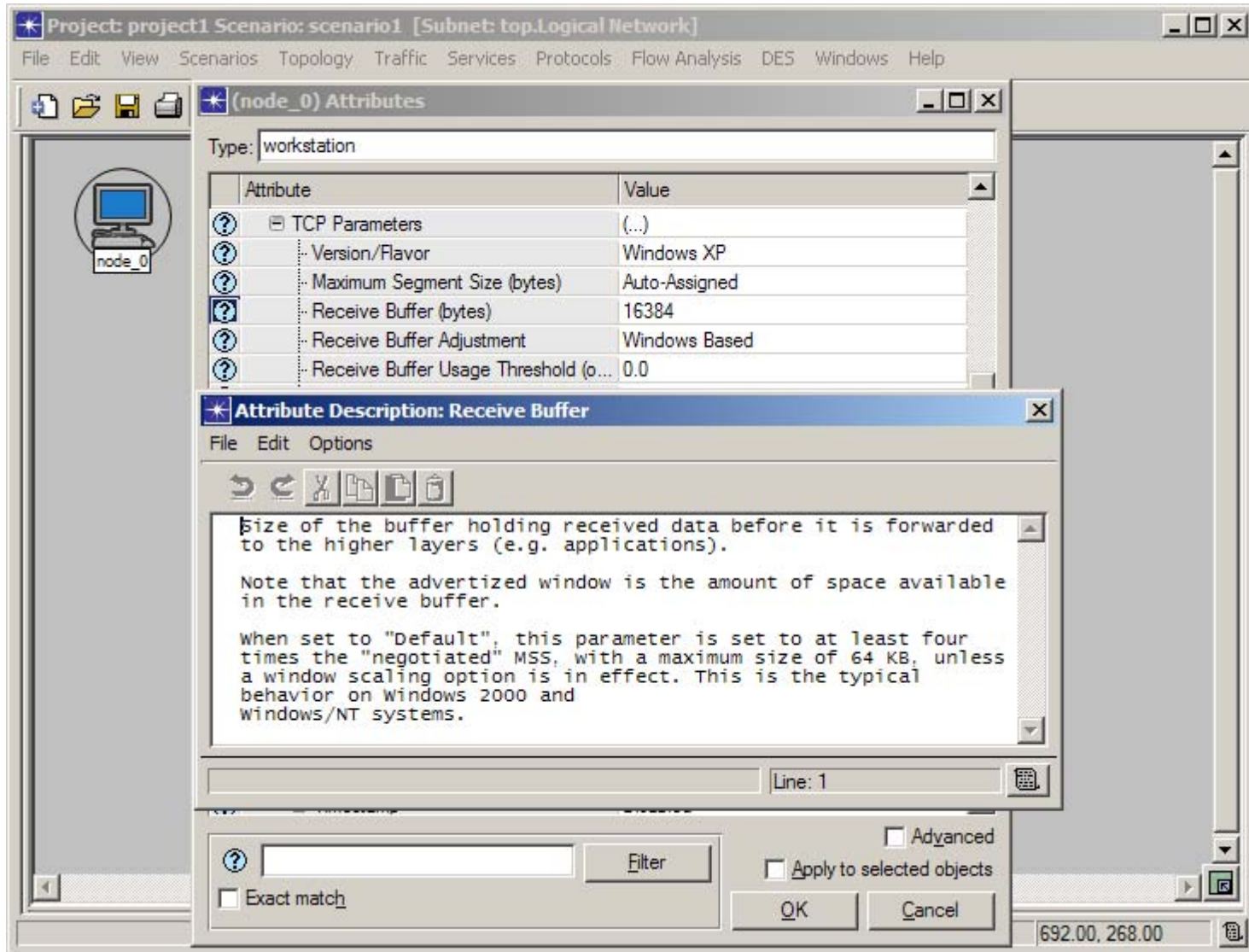
- Ethernet II, Src: Notebook_d3:25:19 (00:06:1b:d3:25:19), Dst: All-MSRP-routers_34 (00:00:0c:07:ac:34)
- Internet Protocol, Src: 130.230.52.139 (130.230.52.139), Dst: 217.70.129.242 (217.70.129.242)
- Transmission Control Protocol, Src Port: optima-vnet (1051), Dst Port: http (80), Seq: 0, Window size: 65535
 - Source port: optima-vnet (1051)
 - Destination port: http (80)
 - Sequence number: 0 (relative sequence number)
 - Header length: 28 bytes
 - Flags: 0x02 (SYN)
 - Checksum: 0x4337 [correct]
 - Options: (8 bytes)
 - Maximum segment size: 1460 bytes
 - NOP
 - NOP
 - SACK permitted

Packet bytes:

```
0000 00 00 0c 07 ac 34 00 06 1b d3 25 19 08 00 45 00  ....4.. ..%...E.
0010 00 30 21 be 40 00 80 06 c6 5f 82 e6 34 8b d9 46  .0!.@... ..4..F
0020 81 f2 04 1b 00 50 84 70 a4 62 00 00 00 00 70 02  ....P.p .b....p.
0030 ff ff 43 37 00 00 02 04 05 b4 01 01 04 02      ..c7.... .....
```

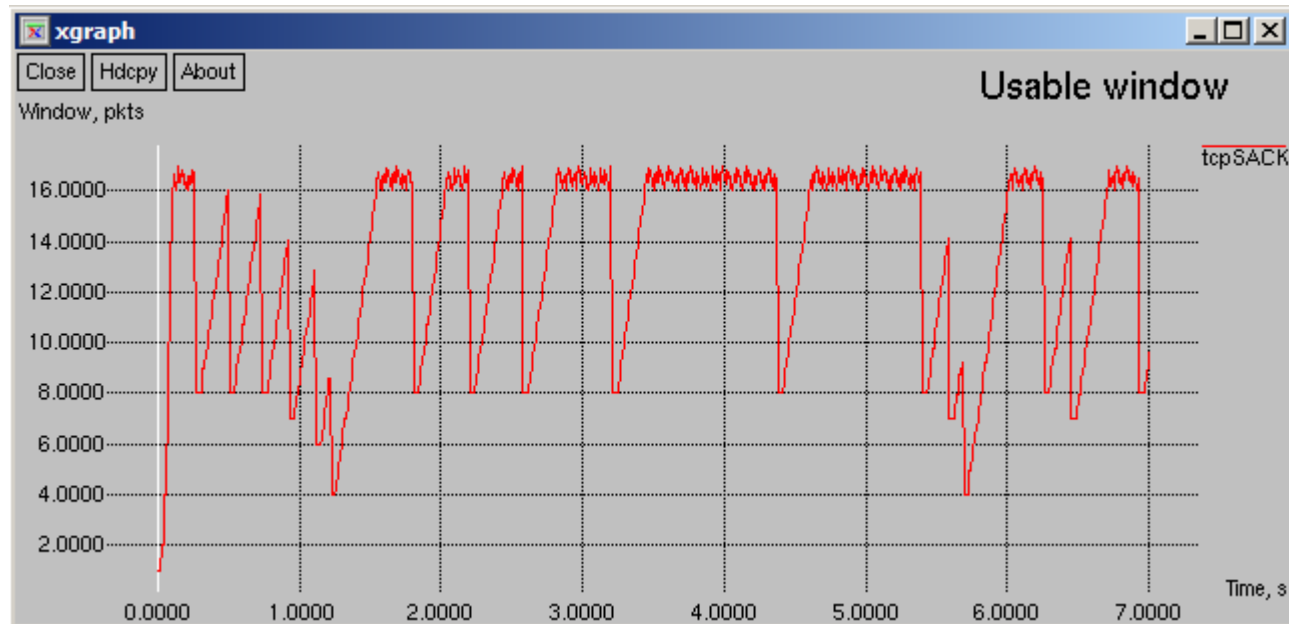
Background (cont'd)

⌘ OPNET Modeler 14.0.A PL3 (Object: ethernet_wkstn)



Background (cont'd)

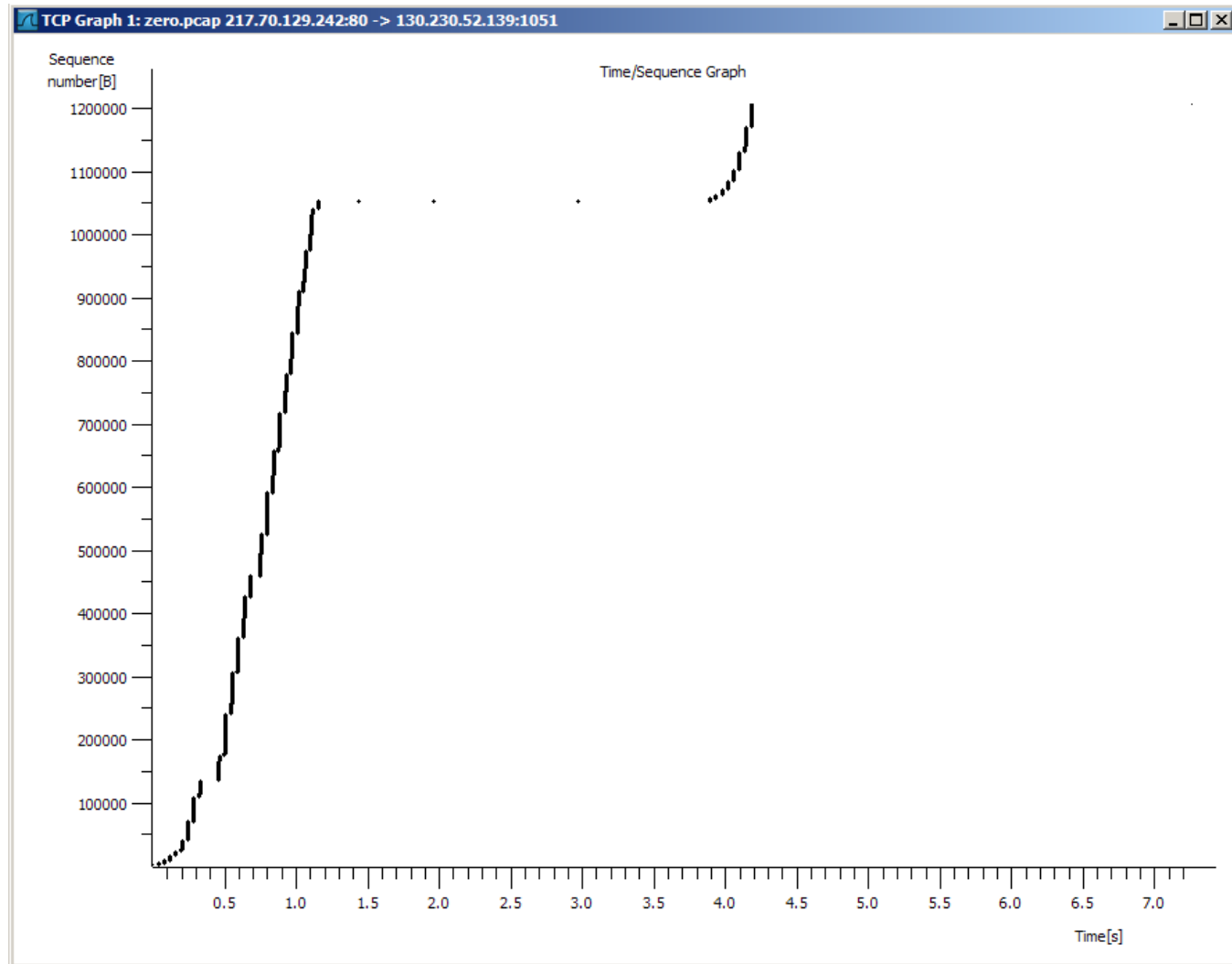
- ⌘ Sender's usable window = **min (cwnd, rwnd)**
- ⌘ Most analytical models use one of the following assumptions about endpoints:
 - A. The TCP sender is only constrained by the congestion window (cwnd)
 - ⊗ I.e., the receive buffer has infinite buffering capacity ($rwnd = \infty$, $cwnd < rwnd$)
 - B. During the loss-free period, the congestion window grows up to the receive window size and then stays constant and equal to this value
 - ⊗ I.e., the receiving application extracts bytes from the buffer instantly ($rwnd = \text{const}$)
- ⌘ In ns2, there is no dynamic window advertisement, too



Background (cont'd)

⌘ Captured by the Wireshark network protocol analyzer (1.15 MB file download)

☒ What's up?



Background (cont'd)

- ⌘ At 1.201527 s of the TCP connection, the data flow was interrupted from the receiver side by setting rwnd to zero
- ⌘ At 3.895820 s of the TCP connection, the data flow was resumed by announcing nonzero rwnd (rwnd = 4600 bytes)
- ⌘ Thus, actual receive window size is adjusted dynamically according to the available buffer space

Time	Source	Destination	Protocol	Info
1.159744	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]
1.159780	130.230.52.139	217.70.129.242	TCP	optima-vnet > http [ACK] Seq=584 Ack=1023461 win=27575 Len=0
1.159866	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]
1.159991	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]
1.160027	130.230.52.139	217.70.129.242	TCP	optima-vnet > http [ACK] Seq=584 Ack=1026381 win=24655 Len=0
1.160083	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]
1.160237	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]
1.160273	130.230.52.139	217.70.129.242	TCP	optima-vnet > http [ACK] Seq=584 Ack=1029301 win=21735 Len=0
1.160329	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]
1.160484	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]
1.160528	130.230.52.139	217.70.129.242	TCP	optima-vnet > http [ACK] Seq=584 Ack=1032221 win=18815 Len=0
1.160584	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]
1.160730	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]
1.160767	130.230.52.139	217.70.129.242	TCP	optima-vnet > http [ACK] Seq=584 Ack=1035141 win=15895 Len=0
1.160822	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]
1.160999	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]
1.161037	130.230.52.139	217.70.129.242	TCP	optima-vnet > http [ACK] Seq=584 Ack=1038061 win=12975 Len=0
1.161052	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]
1.199320	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]
1.199417	130.230.52.139	217.70.129.242	TCP	optima-vnet > http [ACK] Seq=584 Ack=1040321 win=10715 Len=0
1.199449	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]
1.199512	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]
1.199546	130.230.52.139	217.70.129.242	TCP	optima-vnet > http [ACK] Seq=584 Ack=1043241 win=7795 Len=0
1.199597	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]
1.201016	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]
1.201065	130.230.52.139	217.70.129.242	TCP	optima-vnet > http [ACK] Seq=584 Ack=1045361 win=5675 Len=0
1.201123	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]
1.201262	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]
1.201298	130.230.52.139	217.70.129.242	TCP	optima-vnet > http [ACK] Seq=584 Ack=1048281 win=2755 Len=0
1.201352	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]
1.201492	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]
1.201527	130.230.52.139	217.70.129.242	TCP	[TCP zerowindow] optima-vnet > http [ACK] Seq=584 Ack=1051036 win=0 Len=0
1.483208	217.70.129.242	130.230.52.139	TCP	[TCP Keep-Alive] http > optima-vnet [ACK] Seq=1051035 Ack=584 win=6996 Len=0
1.483288	130.230.52.139	217.70.129.242	TCP	[TCP zerowindow] optima-vnet > http [ACK] Seq=584 Ack=1051036 win=0 Len=0
2.007222	217.70.129.242	130.230.52.139	TCP	[TCP Keep-Alive] http > optima-vnet [ACK] Seq=1051035 Ack=584 win=6996 Len=0
2.007295	130.230.52.139	217.70.129.242	TCP	[TCP zerowindow] optima-vnet > http [ACK] Seq=584 Ack=1051036 win=0 Len=0
3.015228	217.70.129.242	130.230.52.139	TCP	[TCP Keep-Alive] http > optima-vnet [ACK] Seq=1051035 Ack=584 win=6996 Len=0
3.015295	130.230.52.139	217.70.129.242	TCP	[TCP zerowindow] optima-vnet > http [ACK] Seq=584 Ack=1051036 win=0 Len=0
3.895820	130.230.52.139	217.70.129.242	TCP	[TCP window update] optima-vnet > http [ACK] Seq=584 Ack=1051036 win=4600 Len=0
3.936388	217.70.129.242	130.230.52.139	TCP	[TCP segment of a reassembled PDU]

Background (cont'd)

- ⌘ As you can see, the default value of the maximum TCP receive window size is different in documentation/OS/simulator
- ⌘ Moreover, receive window advertisement is dynamic, not static
 - ☒ But since we are focusing on TCP performance rather than general client-server performance, we do not model sender or receiver delays due to scheduling or buffering limitations
 - ☒ Instead, we assume that for the duration of the data transfer, the sender sends full-sized segments (packets) as fast as its congestion window allows, and the receiver advertises a consistent receive window (i.e., $rwnd = \text{const}$)

- ⌘ Summary
- ⌘ **Testing is critical**
- ⌘ Testing TCP parameters and algorithms can supply valuable information for a number of purposes:
 - ☒ Identifying reasons for poor TCP performance
 - ☒ Validation of simulations and analytical models
 - ☒ Evaluating security threats
 - ☒ Comparing features of different TCP software implementation
 - ☒ Network monitoring

Related Work

- ⌘ TCP performance modeling has received a lot of attention during the last decade
 - ⊞ Firstly, with the help of analytical models it is possible to evaluate performance of TCP data transfers and TCP-friendliness of multimedia services
 - ⊞ Secondly, analytical models can help researchers to evaluate the existing TCP implementations and to make design decisions about novel TCP mechanisms

- ⌘ Jörgen Olsén. **“Stochastic modeling and simulation of the TCP protocol”**. PhD thesis, Uppsala University, Sweden, October 2003

- ⌘ The first part of the thesis introduces TCP and contains an extensive TCP modeling survey that summarizes the most important TCP modeling work
- ⌘ Reviewed models are categorized as:
 - ⊞ Renewal theory models
 - ⊞ Fixed-point methods
 - ⊞ Fluid models
 - ⊞ Processor sharing models
 - ⊞ Theoretic sharing models
- ⌘ The advantages and disadvantages of each category are discussed and guidelines for future work are given

Modeling Connection Establishment

⌘ Related work

⌘ N. Cardwell, S. Savage, and T. Anderson. “**Modeling TCP latency**”. In Proc. of IEEE INFOCOM 2000, March 2000, pp. 1742-1751

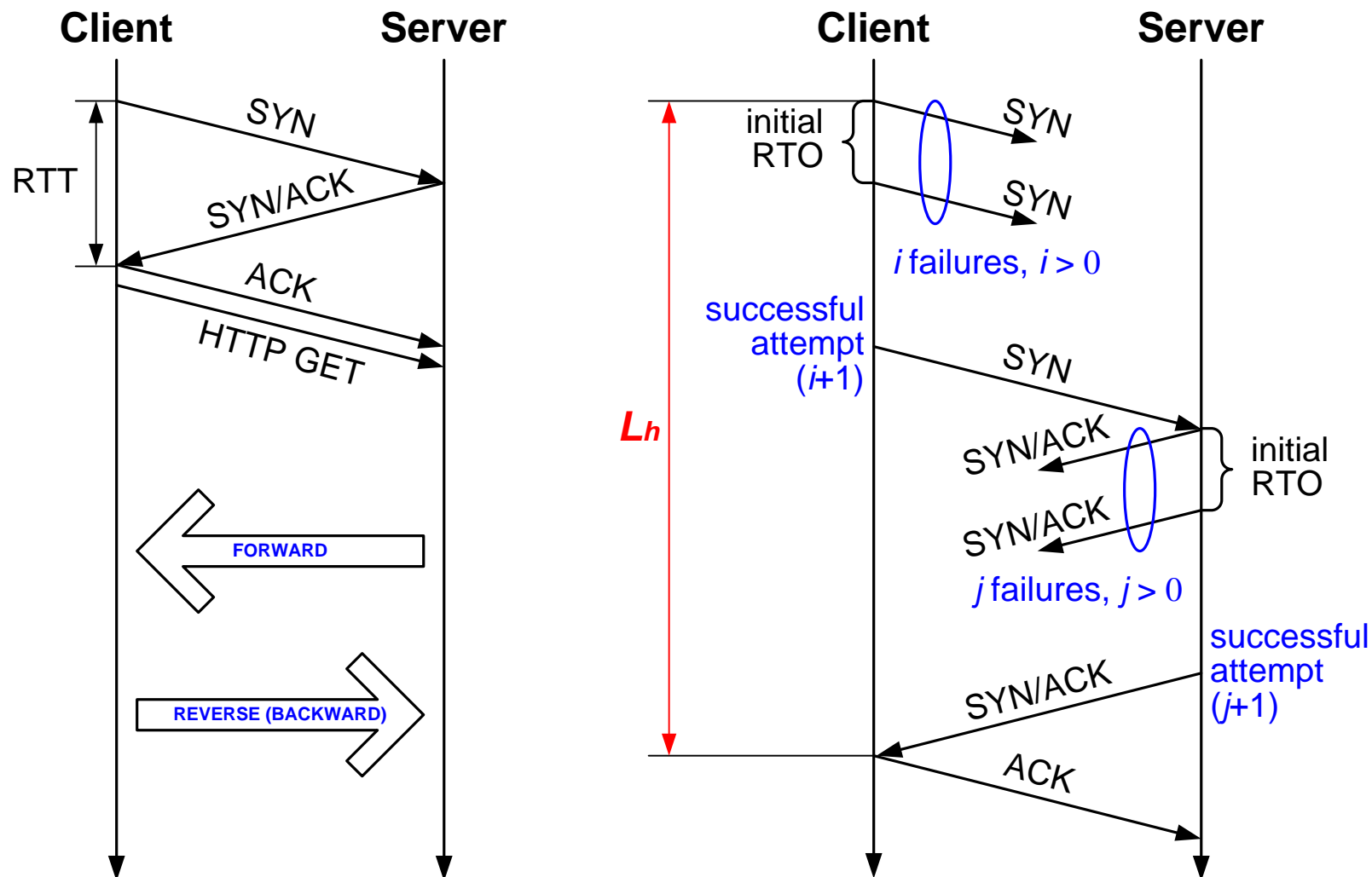
- ☑ This paper provides an expression for the expected value of time required for the connection establishment handshake that begins a TCP connection

Connection Establishment (cont'd)

⌘ Three-way handshake

- ☒ The initiating host, typically the client, performs an active open by sending a SYN segment with its initial sequence number, x
- ☒ The server performs a passive open; when it receives a SYN segment it replies with a SYN segment of its own, containing its initial sequence number, y , as well as an ACK for the active opener's initial sequence number
- ☒ When the active opener receives this SYN/ACK packet, it knows that the connection has been successfully established. It confirms this by sending an ACK of the passive opener's initial sequence number
- ☒ At each stage of this process, if either party does not receive the ACK that it is expecting within some retransmission timeout (RTO), initially 3 seconds (RFC 1122), it retransmits its SYN and waits twice as long for a response
- ☒ This doubling (known as "exponential backoff") is repeated for each unsuccessful retransmission
- ☒ Most TCP implementations abort connection establishment attempts after 4-6 failures

Connection Establishment (cont'd)



Connection Establishment (cont'd)

p_f - "forward" packet loss rate along the path from passive opener to active opener;

p_r - "reverse" packet loss rate along the path from active opener to passive opener;

$P_h(i, j)$ - the probability of having a three-way handshake consisting of exactly i failures transmitting SYNs, followed by one successful SYN, followed by exactly j failures transmitting SYN/ACKs, followed by one successful SYN/ACK;

$$P_h(i, j) = p_r^i (1 - p_r) \cdot p_f^j (1 - p_f)$$

The latency, $L_h(i, j)$, for this process is

$$L_h(i, j) = RTT + \left(\sum_{k=0}^{i-1} 2^k RTO \right) + \left(\sum_{k=0}^{j-1} 2^k RTO \right) =$$

$$\boxed{\sum_{k=m}^n x^k = \frac{x^{n+1} - x^m}{x-1} \Rightarrow \sum_{k=0}^{n-1} 2^k = 2^n - 1}$$

$$= RTT + (2^i - 1)RTO + (2^j - 1)RTO = RTT + (2^i + 2^j - 2)RTO$$

Connection Establishment (cont'd)

The probability that L_h is t seconds or less is

$$P[L_h \leq t] = \sum_{L_h(i,j) \leq t} P_h(i,j)$$

The mathematical expectation of the overall connection establishment latency is

$$\begin{aligned} E[L_h] &= RTT + \sum_{i=0}^{\infty} L_h(i) P_h(i) + \sum_{j=0}^{\infty} L_h(j) P_h(j) = && \boxed{\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}, \quad |x| < 1} \\ RTT + \sum_{i=0}^{\infty} p_r^i (1-p_r) \cdot (2^i - 1) RTO + \sum_{j=0}^{\infty} p_f^j (1-p_f) \cdot (2^j - 1) RTO &= \\ RTT + RTO(1-p_r) \sum_{i=0}^{\infty} \left((2p_r)^i - p_r^i \right) + RTO(1-p_f) \sum_{j=0}^{\infty} \left((2p_f)^j - p_f^j \right) &= \\ RTT + RTO(1-p_r) \left[\sum_{i=0}^{\infty} (2p_r)^i - \sum_{i=0}^{\infty} p_r^i \right] + RTO(1-p_f) \left[\sum_{j=0}^{\infty} (2p_f)^j - \sum_{j=0}^{\infty} p_f^j \right] &= \\ RTT + RTO \left(\frac{1-p_r}{1-2p_r} - 1 \right) + RTO \left(\frac{1-p_f}{1-2p_f} - 1 \right) &= RTT + RTO \left(\frac{1-p_r}{1-2p_r} + \frac{1-p_f}{1-2p_f} - 2 \right) \end{aligned}$$

Modeling Slow Start Phase

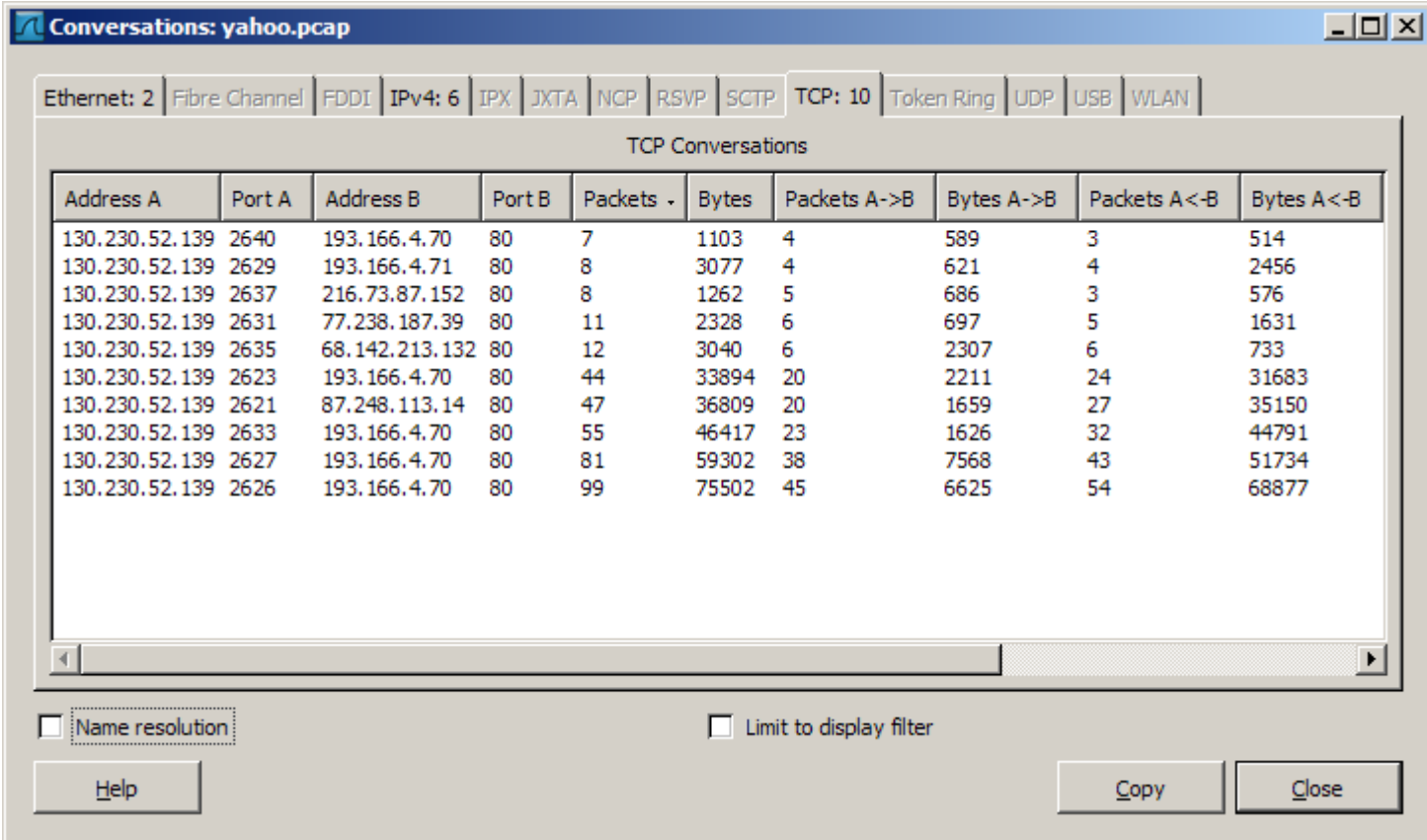
- ⌘ The “mice and elephants” phenomenon
- ⌘ Much of the traffic on the Internet is carried by a small number of large flows (**elephants**), while most of the flows are short in duration and carry a small amount of traffic (**mice**)
 - ☒ TCP was originally designed for elephants, where a sender probes for the end-to-end available bandwidth in a slow start phase, and then spends most of its data transfer operation in a congestion avoidance phase
- ⌘ **TCP traffic is dominated by mice** (also known as short-lived flows or short transfers)
 - ☒ I.e., flows that are short in duration and carry a small amount of traffic
 - ☒ Those flows are short enough to experience any losses and spend the most part of their life-time in the initial slow start phase
- ⌘ Thus, it is important to model TCP performance taking into account TCP startup mechanisms such as connection establishment phase and the initial slow start phase

Modeling Slow Start (cont'd)

⌘ Results of an HTTP request for **Yahoo!** home page

☑ Captured by the Wireshark network protocol analyzer, March 10, 2008

☑ **TOTAL:** 10 TCP connections; 372 packets; 262,734 bytes



The screenshot shows the 'Conversations: yahoo.pcap' window in Wireshark. The 'TCP: 10' tab is selected, displaying a table of TCP conversations. The table has columns for Address A, Port A, Address B, Port B, Packets -, Bytes, Packets A->B, Bytes A->B, Packets A<-B, and Bytes A<-B. The data is as follows:

Address A	Port A	Address B	Port B	Packets -	Bytes	Packets A->B	Bytes A->B	Packets A<-B	Bytes A<-B
130.230.52.139	2640	193.166.4.70	80	7	1103	4	589	3	514
130.230.52.139	2629	193.166.4.71	80	8	3077	4	621	4	2456
130.230.52.139	2637	216.73.87.152	80	8	1262	5	686	3	576
130.230.52.139	2631	77.238.187.39	80	11	2328	6	697	5	1631
130.230.52.139	2635	68.142.213.132	80	12	3040	6	2307	6	733
130.230.52.139	2623	193.166.4.70	80	44	33894	20	2211	24	31683
130.230.52.139	2621	87.248.113.14	80	47	36809	20	1659	27	35150
130.230.52.139	2633	193.166.4.70	80	55	46417	23	1626	32	44791
130.230.52.139	2627	193.166.4.70	80	81	59302	38	7568	43	51734
130.230.52.139	2626	193.166.4.70	80	99	75502	45	6625	54	68877

At the bottom of the window, there are checkboxes for 'Name resolution' and 'Limit to display filter', both of which are unchecked. There are also 'Help', 'Copy', and 'Close' buttons.

Modeling Slow Start (cont'd)

- ⌘ Results of an HTTP request for **@MAIL.RU** home page
 - ☑ Captured by the WildPackets OmniPeek network protocol analyzer, March 10, 2008
 - ☑ **TOTAL**: 59 TCP connections; 885 packets; 346,557 bytes
 - ☑ **Average** (per connection): 15 packets/connection; 6 KB/connection

The screenshot shows the OmniPeek interface with the following statistics:

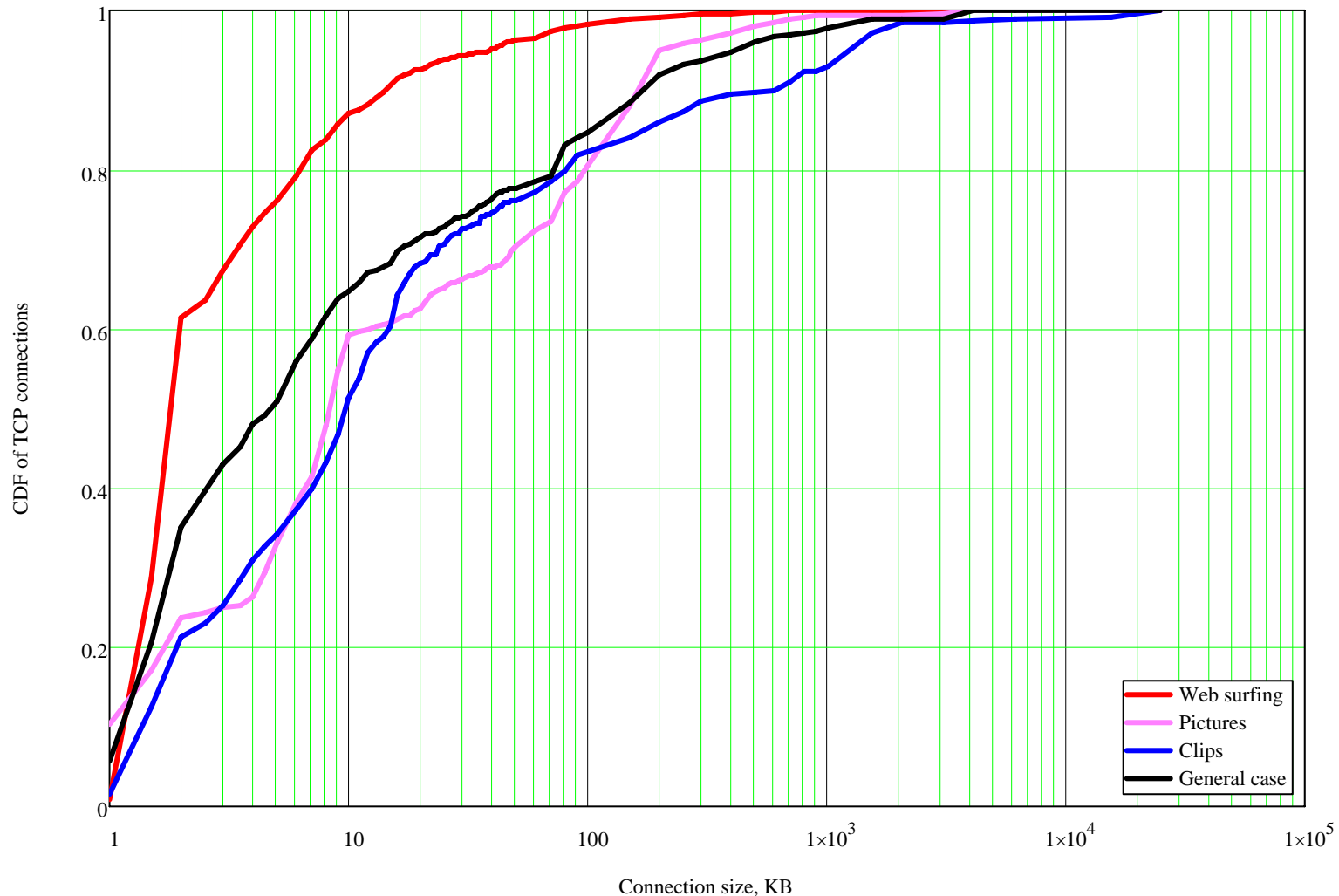
Flows analyzed:	59	Flows recycled:	0
Events detected:	5	Packets dropped:	0

Name	Flows	Packets	Bytes
Web	59	885	346,557
81.19.66.19	1	9	2,041
88.212.196.78	1	12	2,451
194.67.45.129	1	10	1,456
194.67.57.157	2	192	119,845
194.67.57.226	1	22	15,028
194.186.55.125	51	622	202,704
217.73.200.174	2	18	3,032

At the bottom of the window, the status bar shows: Packets: 885, Duration: 0:00:20.

Modeling Slow Start (cont'd)

⌘ Captured by the Colasoft Capsa network protocol analyzer, April 01, 2004

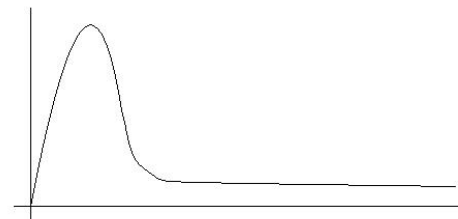


Modeling Slow Start (cont'd)

⌘ Heavy-tailed distribution

⌘ Many of the distributions shown in today's Internet have the property of being **heavy-tailed**, e.g.:

- ⊞ Length of individual flows
- ⊞ User think time
- ⊞ Size of available files on the Web



⌘ A distribution is said to be heavy-tailed if the asymptotic shape of the distribution is power-law with exponent less than 2 regardless of the behavior of the distribution for small values of the random variable, i.e., the probability that a random variable is larger than x is

$$P[X > x] \sim x^{-\alpha}, \quad \text{as } x \rightarrow \infty, \quad 0 < \alpha < 2$$

⌘ The reason that such distributions are called “heavy-tailed” is that, compared to those more commonly used distributions such as exponential and normal distributions, a random variable that follows a heavy-tailed distribution can give rise to extremely large values with non-negligible probability

Modeling Slow Start (cont'd)

⌘ Slow start (RFC 2581)

- ☒ After the connection establishment phase, the initial slow start phase begins
- ☒ During this phase, the sender increments its **congestion window (cwnd)** by one segment for each acknowledgement (ACK) received that acknowledges new data
- ☒ The upper bound for the **initial window (IW)** (RFC 3390) is given by

$\min(4 \cdot \text{MSS}, \max(2 \cdot \text{MSS}, 4380 \text{ bytes}))$

- ☒ Thus, the sender may use a 1, 2, 3, or 4 segment initial window, provided the combined size of the segments does not exceed 4380 bytes
- ☒ Slow start ends when cwnd exceeds (or, optionally, when it reaches it) **slow start threshold (ssthresh)** or when congestion is observed

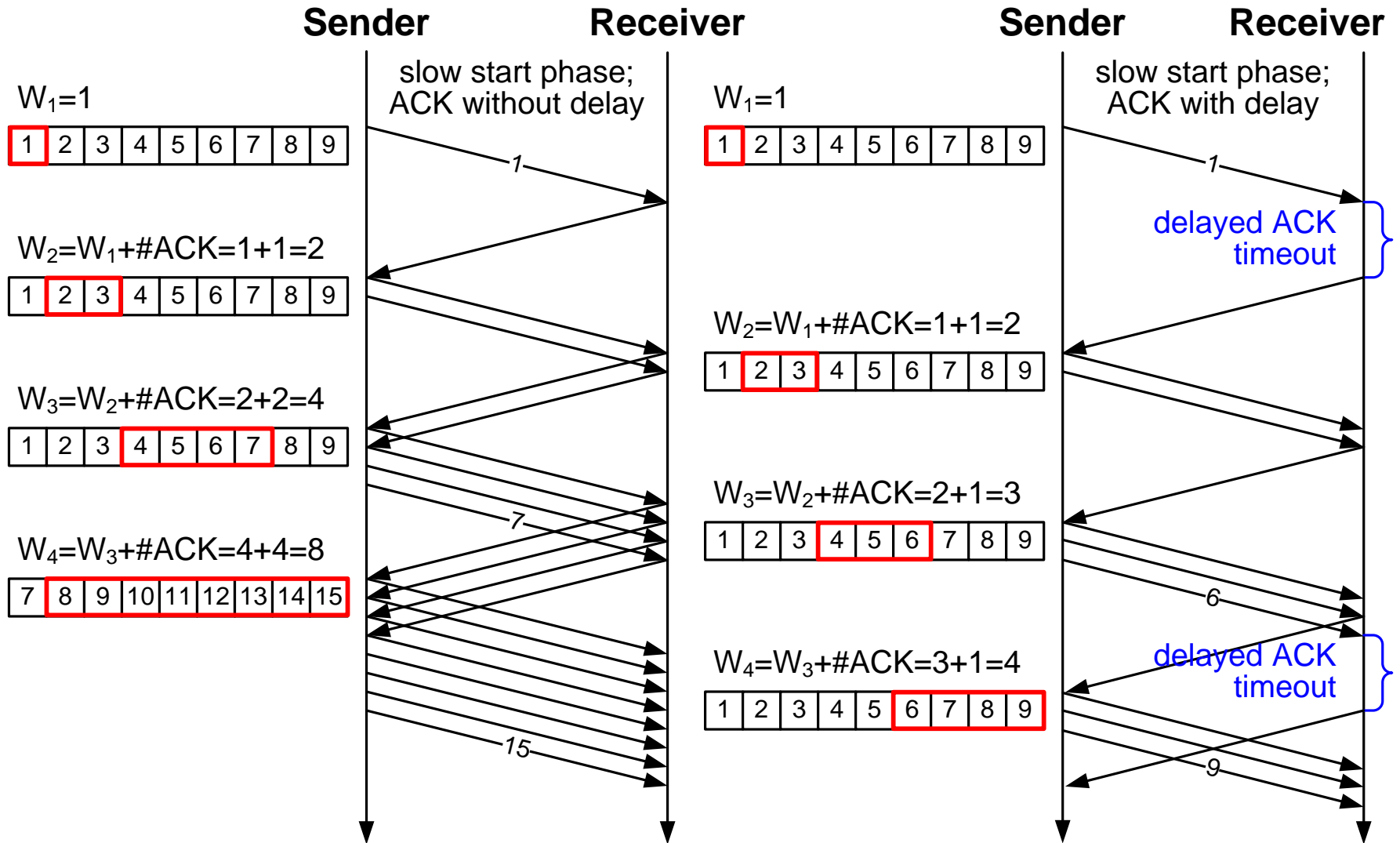
⌘ Delayed ACK algorithm (RFC 1122)

- ☒ The receiver sends one ACK for every two full-sized segments that it gets or if the delayed ACK timeout expires

⌘ Assumptions

- ☒ TCP behavior is considered in terms of “rounds”, where a round starts when the sender begins the transmission of a window of segments and ends when the sender receives an acknowledgment for one or more of these segments
- ☒ It is assumed that the time to send all the segments in a window is smaller than the duration of a round and that the duration of a round is independent of the window size

Modeling Slow Start (cont'd)



Modeling Slow Start (cont'd)

⌘ Related work

- ⌘ N. Cardwell, S. Savage, and T. Anderson. **"Modeling TCP latency"**. In Proc. of IEEE INFOCOM 2000, March 2000, pp. 1742-1751
- ⌘ D. Zheng, G. Y. Lazarou, and R. Hu. **"A stochastic model for short-lived TCP flows"**. In Proc. of ICC 2003, volume 26, May 2003, pp. 76-81
- ⌘ B. Sikdar, S. Kalyanaraman, and K. S. Vastola. **"Analytic models for the latency and steady-state throughput of TCP Tahoe, Reno and SACK"**. IEEE/ACM Transactions on Networking, volume 11, issue 6, December 2003, pp. 959-971
 - ☒ *D. Phillips, J. Hu, B. Lloyd Smith, R. Harris. "A note on an analytic model for slow start in TCP". In Proc. of the 11th IEEE International Conference on Networks, September 2003, pp. 261-264*

⌘ These papers provide analytical models for:

- ☒ Congestion window increase pattern
- ☒ Total number of transmitted segments within a given number of slow start rounds
- ☒ Number of slow start rounds to transmit a given number of segments
- ☒ Congestion window size TCP achieves after sending a given number of segments in unconstrained slow start phase

Model by N. Cardwell et al.

- ⌘ During slow start, each round the sender sends as many data segments as its cwnd allows
- ⌘ The receiver sends one ACK for every b -th data segment that it receives
 - ☒ $b = 1$ if the receiver immediately acknowledges every segment that it gets (no delay)
 - ☒ $b = 2$ if the receiver sends one ACK for every two full-sized segments that it gets (delayed acknowledgement algorithm)
- ⌘ Then each round the sender will get approximately (W/b) ACKs
- ⌘ Because the sender is in slow start, for each new ACK it receives, it increases its cwnd by one segment
- ⌘ Thus,

W_i - the sender's cwnd at the beginning of round i ;

γ - the rate of exponential growth of cwnd during slow start;

$$W_{i+1} = W_i + W_i/b = (1 + 1/b) \cdot W_i = \gamma \cdot W_i$$

Model by N. Cardwell et al. (cont'd)

If the sender starts with an initial window of W_1 segments, then $ssdata_n$, the amount of data sent by the end of slow start round n , can be closely approximated by a geometric series as

$$ssdata_n = W_1 + W_1 \cdot \gamma + W_1 \cdot \gamma^2 + \dots + W_1 \cdot \gamma^{n-1} = W_1 \cdot \sum_{k=1}^n \gamma^{k-1} =$$

$$\sum_{k=m}^n x^k = \frac{x^{n+1} - x^m}{x-1}$$

$$= \frac{W_1}{\gamma} \cdot \sum_{k=1}^n \gamma^k = \frac{W_1}{\gamma} \cdot \frac{\gamma^{n+1} - \gamma}{\gamma - 1} = \frac{W_1}{\gamma} \cdot \frac{\gamma \cdot (\gamma^n - 1)}{\gamma - 1} = W_1 \cdot \frac{\gamma^n - 1}{\gamma - 1}$$

The number of slow start rounds to transfer $ssdata_n$ segments of data is

$$n = \log_{\gamma} \left(\frac{ssdata_n \cdot (\gamma - 1)}{W_1} + 1 \right)$$

The window TCP achieves after sending d segments in unconstrained slow start is

$$W_{SS}(d) = W_1 \cdot \gamma^{n-1} = W_1 \cdot \left(\frac{d \cdot (\gamma - 1)}{W_1} + 1 \right) \cdot \gamma^{-1} = \frac{d \cdot (\gamma - 1)}{\gamma} + \frac{W_1}{\gamma}$$

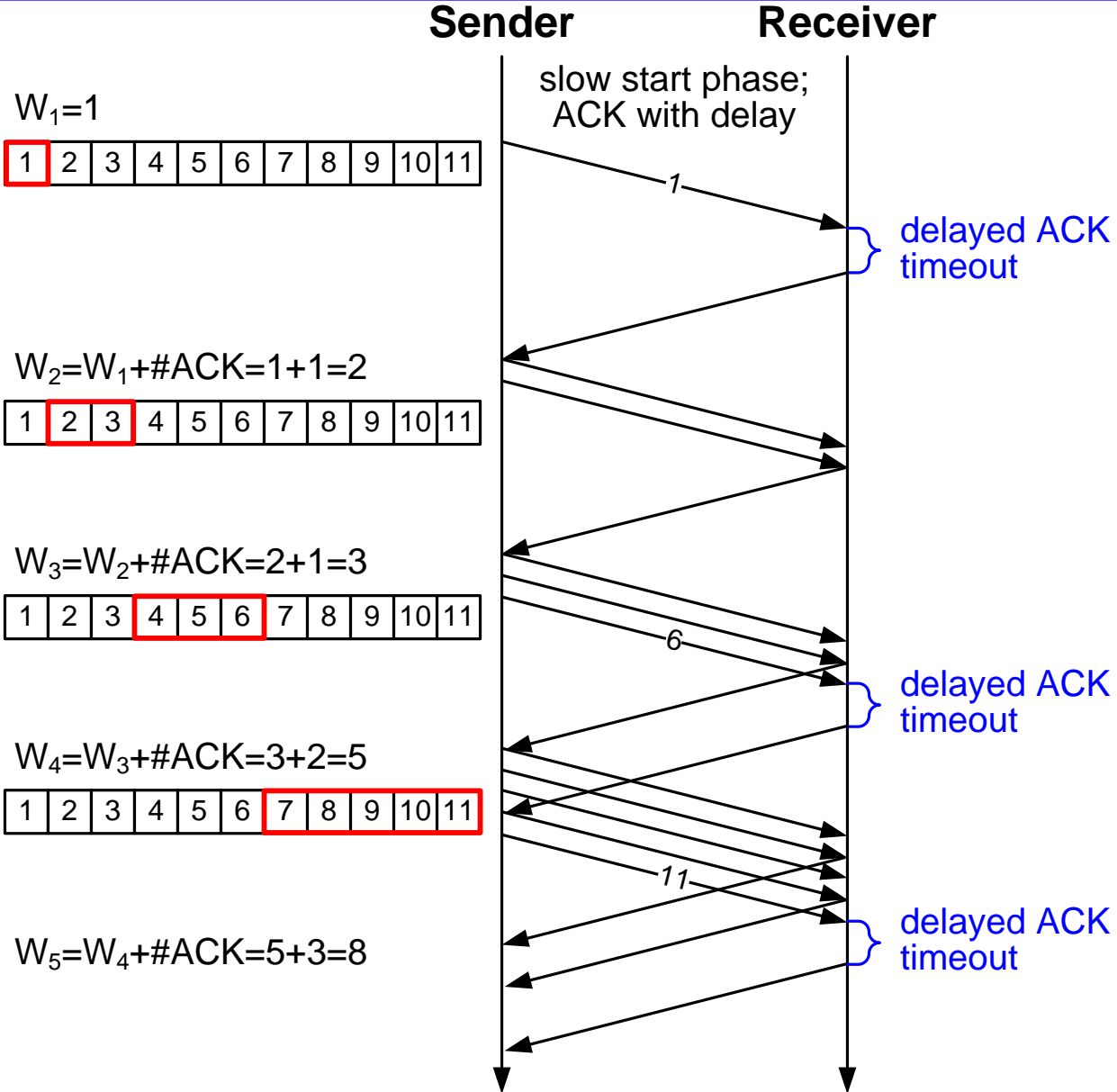
Model by N. Cardwell et al. (cont'd)

- ⌘ Given typical parameters of $\gamma = 1.5$ ($b = 2$) and $1 \leq IW \leq 3$ (MSS = 1460 bytes), it follows that

$$W_{ss}(d) = \frac{d \cdot (\gamma - 1)}{\gamma} + \frac{W_1}{\gamma} \approx \frac{d}{3}$$

- ⌘ **Thus, to reach any congestion window, W , a flow needs to send approximately $3W$**
- ⌘ Given initial ssthresh of 65,535 bytes (RFC 2001), it is easy to see why many TCP flows spend most of their lifetimes in the initial slow start phase
 - ☑ To reach ssthresh of 65,535 bytes, a TCP flow needs to send approximately 192 KB of data

Model by D. Zheng et al.



Model by D. Zheng et al. (cont'd)

W_i - the sender's cwnd at the beginning of round i ;

γ - the rate of exponential growth of cwnd during slow start;

$\lceil x \rceil$ - the smallest integer bigger than x (the ceiling function);

$$W_{i+1} = W_i + \left\lceil \frac{1}{2} \cdot W_i \right\rceil = W_i + \left\lceil \frac{1}{2} \cdot \left(W_{i-1} + \left\lceil \frac{1}{2} \cdot W_{i-1} \right\rceil \right) \right\rceil = W_i + \left\lceil \frac{1}{2} \cdot \left\lceil \frac{3}{2} \cdot W_{i-1} \right\rceil \right\rceil \approx$$

$\approx W_i + W_{i-1} \Rightarrow$ the Fibonacci sequence

$$F(n) := \begin{cases} 0, & \text{if } n = 0; \\ 1, & \text{if } n = 1; \\ F(n-1) + F(n-2), & \text{if } n > 1. \end{cases}$$

Thus, for $W_1 = 1$ it follows

$$W_{i+1} = W_i + W_{i-1} \Rightarrow 1, 2, 3, 5, 8, 13, 21, \dots$$

vs.

$$W_{i+1} = \gamma \cdot W_i = \frac{3}{2} \cdot W_i \Rightarrow 1, 1.5, 2.25, 3.38, 5.06, 7.59, 11.39, 17.09, \dots$$

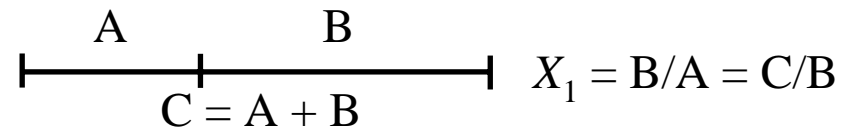
Model by D. Zheng et al. (cont'd)

$$W_n = C_1 \cdot X_1^n + C_2 \cdot X_2^n, \quad n = 1, 2, 3, \dots$$

$$X_1 = \frac{1 + \sqrt{5}}{2};$$

X_1 - the golden ratio (number), $X_1 = 1.6180339887498948482045868343656$;

$$X_2 = \frac{1 - \sqrt{5}}{2};$$

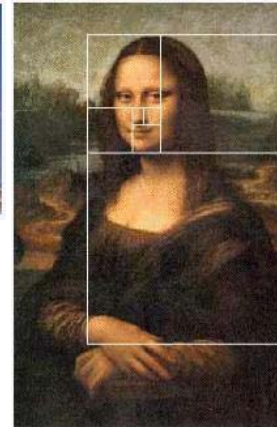
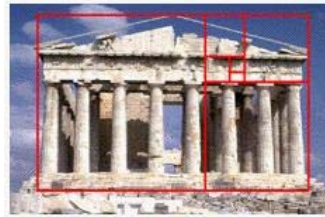


$$C_1 + C_2 = 1$$

For $W_1 = 1$ it follows

$$\begin{cases} 1 = C_1 \cdot \frac{1 + \sqrt{5}}{2} + C_2 \cdot \frac{1 - \sqrt{5}}{2} \\ C_1 + C_2 = 1 \end{cases}$$

$$C_1 = \frac{5 + \sqrt{5}}{10}, \quad C_2 = \frac{5 - \sqrt{5}}{10}$$



Model by D. Zheng et al. (cont'd)

The number of segments sent by the end of slow start round n , can be approximated as follows

$$ssdata_n = \sum_{k=1}^n W_k = \sum_{k=1}^n F(k) =$$

$$\begin{aligned} F(0) + F(1) + F(2) + F(3) + \dots + F(n) &= F(n+2) - 1 \\ 0 + 1 + \boxed{1 + 2 + 3 + 5 + 8 + \dots + F(n)} &= F(n+2) - 1 \\ 1 + 2 + 3 + 5 + 8 + \dots + F(n) &= F(n+2) - 2 \end{aligned}$$

$$= F(n+2) - 2 = C_1 \cdot X_1^{n+2} + C_2 \cdot X_2^{n+2} - 2 \approx C_1 \cdot X_1^{n+2} - 2$$

$$\begin{aligned} n = 1 &\Rightarrow C_2 \cdot X_2^{n+2} = \frac{5 - \sqrt{5}}{10} \cdot \left(\frac{1 - \sqrt{5}}{2} \right)^3 = -0.065 \\ n = 2 &\Rightarrow C_2 \cdot X_2^{n+2} = 0.04 \\ &\dots \end{aligned}$$

Model by D. Zheng et al. (cont'd)

The number of slow start rounds to transfer $ssdata_n$ segments of data is

$$n = \log_{X_1} \left(\frac{ssdata_n + 2}{C_1} \right) - 2$$

The window TCP achieves after sending d segments in unconstrained slow start is

$$W_{SS}(d) = C_1 \cdot X_1^n + C_2 \cdot X_2^n = \frac{d+2}{X_1^2} + C_2 \cdot X_2^n \approx \frac{d+2}{X_1^2}$$

Model by B. Sikdar et al.

- ⌘ B. Sikdar et al. noted that cwnd increase pattern is different in practice and can make significant differences for short-lived flows
- ⌘ To account for such complex behavior of the cwnd, the proposed model uses an averaged pattern which tries to model the expected value of the cwnd for any round



Without delayed ACK timer expiration



With delayed ACK timer expiration



Proposed model

cwnd increase patterns

Model by B. Sikdar et al. (cont'd)

$\lfloor x \rfloor$ - the biggest integer smaller than x (the floor function)

The number of segments transmitted in the n -th round is given by

$$W_n = \left\lfloor 2^{\frac{n-1}{2}} + 2^{\frac{n-2}{2}} \right\rfloor$$

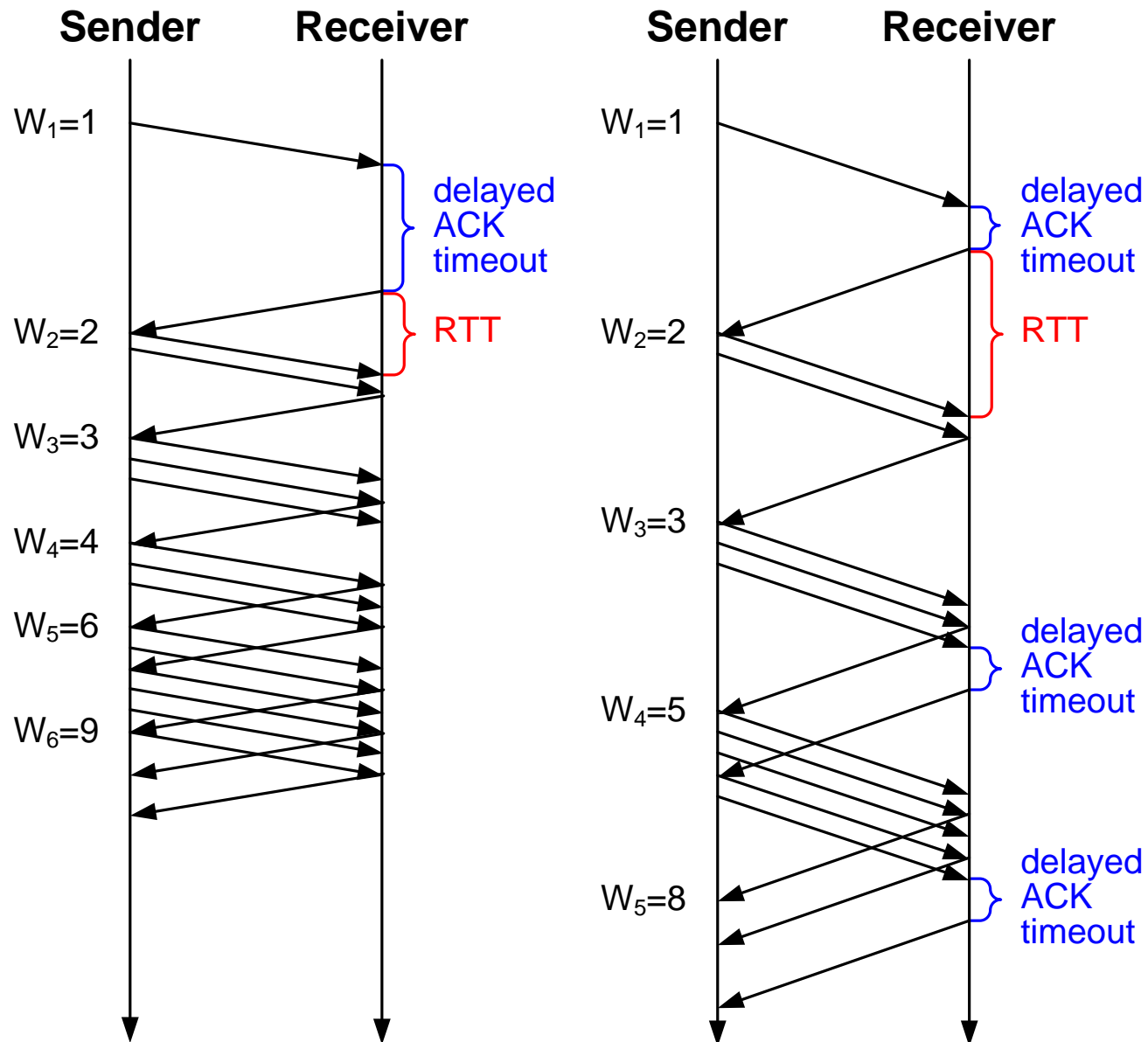
The number of segments sent by the end of slow start round n , can be found as

$$ssdata_n = \sum_{k=1}^n W_k = \left\lfloor 2^{\frac{n+1}{2}} + 3 \cdot 2^{\frac{4n-3}{8}} - 2 - \frac{3\sqrt{2}}{2} \right\rfloor$$

The number of rounds required to transmit $ssdata_n$ segments of data is

$$n = \left\lceil 2 \cdot \log_2 \left(\frac{2 \cdot ssdata_n + 4 + 3\sqrt{2}}{2\sqrt{2} + 3 \cdot 2^{\frac{5}{8}}} \right) \right\rceil$$

Proposed Model



Proposed Model (cont'd)

cwnd increase patterns (in segments)

Round, i	RTT belongs to						
	$(0, T_d]$	$(T_d, 2T_d]$	$2T_d$	$(2T_d, 3T_d]$	$(3T_d, 4T_d]$	$(4T_d, 5T_d]$	$(5T_d, 6T_d]$
1)	1	1	1	1	1	1	1
2)	2	2	2	2	2	2	2
3)	3	3	3	3	3	3	3
4)	4	5	5	5	5	5	5
5)	6	7	7	8	8	8	8
6)	9	10	11	12	12	12	12
7)	13	15	16	18	18	18	18
8)	19	22	24	27	27	27	27
9)	28	33	36	40	41	41	41
10)	42	49	54	60	61	62	62
11)	63	73	81	90	91	93	93
12)	94	109	121	135	136	139	140

Proposed Model (cont'd)

The number of transmitted segments

Round, i	RTT belongs to						
	$(0, T_d]$	$(T_d, 2T_d)$	$2T_d$	$(2T_d, 3T_d]$	$(3T_d, 4T_d]$	$(4T_d, 5T_d]$	$(5T_d, 6T_d]$
1)	1	1	1	1	1	1	1
2)	2	3	2	3	3	3	3
3)	6	6	6	6	6	6	6
4)	9	11	11	11	11	11	11
5)	15	17	17	19	19	19	19
6)	24	26	28	31	31	31	31
7)	36	41	43	49	49	49	49
8)	54	62	67	76	76	76	76
9)	81	95	103	115	117	117	117
10)	123	143	157	175	177	179	179
11)	186	215	238	265	267	272	272
12)	279	323	358	400	402	410	412

Proposed Model (cont'd)

- ⌘ The receiver sends one ACK for every b -th data segment that it gets
 - ☒ $b = 1$ if the receiver immediately acknowledges every segment that it gets
 - ☒ $b = 2$ if the receiver sends one ACK for every two full-sized segments that it gets (delayed acknowledgement algorithm)
- ⌘ For each new ACK the sender receives, it increases its cwnd by one segment
 - ☒ $\gamma = 1 + 1/b$ is the rate of exponential growth of cwnd ($\gamma = 2$ or 1.5)
- ⌘ When $RTT \leq 2 \cdot T_{delACK}$, cwnd increase patterns can be closely approximated by the model proposed by N. Cardwell et al.:

$$W_{i+1} = W_i + \frac{W_i}{b} = \gamma \cdot W_i, \quad i = 1, 2, \dots, n$$

- ⌘ But if the ACK timer expires before a new segment arrives (in the cases when $RTT > 2 \cdot T_{delACK}$), cwnd increase pattern can be approximated as

$$W_{i+1} = W_i + \left\lceil \frac{W_i}{b} \right\rceil = \lceil \gamma \cdot W_i \rceil, \quad i = 1, 2, \dots, n$$

Proposed Model (cont'd)

$$W_{i+1} = \begin{cases} \gamma \cdot W_i, & \text{when } RTT \leq 2 \cdot T_{delACK} \\ \lceil \gamma \cdot W_i \rceil, & \text{when } RTT > 2 \cdot T_{delACK} \end{cases}$$

$$\lceil x \rceil = x + r, \quad 0 \leq r < 1$$

$$W_1 = W_1,$$

$$W_2 = W_1 + \lceil W_1/b \rceil = \lceil W_1 \cdot \gamma \rceil = W_1 \cdot \gamma + r_1$$

$$W_3 = \lceil \gamma \cdot (W_1 \cdot \gamma + r_1) \rceil = W_1 \cdot \gamma^2 + r_1 \cdot \gamma + r_2$$

$$W_4 = \lceil \gamma \cdot (W_1 \cdot \gamma^2 + r_1 \cdot \gamma + r) \rceil = W_1 \cdot \gamma^3 + r_1 \cdot \gamma^2 + r_2 \cdot \gamma + r_3$$

⋮

$$W_n = W_1 \cdot \gamma^{n-1} + r_1 \cdot \gamma^{n-2} + \dots + r_{n-2} \cdot \gamma + r_{n-1}$$

Since $\gamma = 3/2$, we have that $r_j, (j = 1, 2, \dots, n-1)$ is a discrete random variable, which takes on a value of 0 or $1/2$

Uniform distribution between these values implies that $E[r] = \frac{0 + 0.5}{2} = \frac{1}{4}$

Proposed Model (cont'd)

The number of segments sent by the end of slow start round n , can be expressed as

$$\begin{aligned} ssdata_n &= W_1 \cdot \sum_{k=1}^n W_k = W_1 \cdot \sum_{k=1}^n \gamma^{k-1} + E[r] \cdot \sum_{k=1}^n (n-k) \cdot \gamma^{k-1} = \\ &= \frac{\gamma^n \cdot (W_1 \cdot (\gamma - 1) + E[r]) - n \cdot E[r] \cdot (\gamma - 1) - (W_1 \cdot (\gamma - 1) + E[r])}{(\gamma - 1)^2} \end{aligned}$$

After transformation of the obtained expression to the canonical form, we get

$$n - \left(\frac{W_1 \cdot (\gamma - 1) + E[r]}{E[r] \cdot (\gamma - 1)} \right) \cdot \gamma^n + \frac{ssdata_n \cdot (\gamma - 1)^2 + W_1 \cdot (\gamma - 1) + E[r]}{E[r] \cdot (\gamma - 1)} = 0$$

This is a transcendental equation, which cannot be solved by an exact method, only by graphical or numerical methods

A transcendental equation is an equation containing a transcendental function (i.e., a function which cannot be expressed in terms of algebra), e.g.,

$$x = a^x \text{ or } x = \sin x$$

Proposed Model (cont'd)

M.A. Eremin. "High degree equations". Arzamas, 2003 (in Russian)

$$x^n + a_1 x^{n-1} + a_2 x^{n-2} + \dots + a_{n-d} x^d + a_{n-q} x^q + f(x) + a_n = 0,$$

$f(x)$ - transcendental function

The equation determinant is given as

$$p = \frac{m^d}{m^q + \frac{a_n + f(m)}{a_{n-q}}}$$

and

$$p = \frac{a_{n-q}}{m^{n-d} + a_1 m^{n-(d+1)} + a_2 m^{n-(d+2)} + \dots + a_{n-d}}$$

The range of values m (and approximate roots of the equation) can be found from a set of inequalities for

1) $p > 0$

2) $p < 0$

Proposed Model (cont'd)

$$n - \left(\frac{W_1 \cdot (\gamma - 1) + E[r]}{E[r] \cdot (\gamma - 1)} \right) \cdot \gamma^n + \frac{ssdata_n \cdot (\gamma - 1)^2 + W_1 \cdot (\gamma - 1) + E[r]}{E[r] \cdot (\gamma - 1)} = 0$$

$$p = \frac{m}{\left(\frac{W_1 \cdot (\gamma - 1) + E[r]}{E[r] \cdot (\gamma - 1)} \right) \cdot \gamma^m};$$

$$1 - \frac{ssdata_n \cdot (\gamma - 1)^2 + W_1 \cdot (\gamma - 1) + E[r]}{E[r] \cdot (\gamma - 1)}$$

$$p = \frac{\frac{ssdata_n \cdot (\gamma - 1)^2 + W_1 \cdot (\gamma - 1) + E[r]}{E[r] \cdot (\gamma - 1)}}{m + \frac{ssdata_n \cdot (\gamma - 1)^2 + W_1 \cdot (\gamma - 1) + E[r]}{E[r] \cdot (\gamma - 1)}};$$

$$m > \log_\gamma \left(\frac{ssdata_n \cdot (\gamma - 1)^2}{W_1 \cdot (\gamma - 1) + E[r]} + 1 \right) \Rightarrow n \approx \left\lceil \log_\gamma \left(\frac{ssdata_n \cdot (\gamma - 1)^2}{W_1 \cdot (\gamma - 1) + E[r]} + 1 \right) \right\rceil$$

Proposed Model (cont'd)

The comprehensive model of the initial slow start phase:

W_1 - the initial window size; $E[r] = 0.25$;

$$W_n = \begin{cases} W_1 \cdot \gamma^{n-1}, & \text{when } RTT \leq 2 \cdot T_{delACK} \\ W_1 \cdot \gamma^{n-1} + E[r] \cdot \frac{\gamma^{n-1} - 1}{\gamma - 1}, & \text{when } RTT > 2 \cdot T_{delACK} \end{cases}$$

$$ssdata_n = \begin{cases} W_1 \cdot \sum_{k=1}^n \gamma^{k-1}, & \text{when } RTT \leq 2 \cdot T_{delACK} \\ W_1 \cdot \sum_{k=1}^n \gamma^{k-1} + E[r] \cdot \sum_{k=1}^n (n-k) \cdot \gamma^{k-1}, & \text{when } RTT > 2 \cdot T_{delACK} \end{cases}$$

$$n(d) = \begin{cases} \log_{\gamma} \left(\frac{d \cdot (\gamma - 1)}{W_1} + 1 \right), & \text{when } RTT \leq 2 \cdot T_{delACK} \\ \left\lceil \log_{\gamma} \left(\frac{d \cdot (\gamma - 1)^2}{W_1 \cdot (\gamma - 1) + E[r]} + 1 \right) \right\rceil, & \text{when } RTT > 2 \cdot T_{delACK} \end{cases}$$

Proposed Model (cont'd)

- ⌘ Simulation setup
- ⌘ network simulator - ns2
- ⌘ Initial window = {1, 2, 3} segments
- ⌘ Delayed ACK timeout = {100, 150, 200} ms
- ⌘ $RTT \leq N \cdot T_{delACK}$, $N = \{1, 2, \dots, 6\}$
- ⌘ Relative error:

$$\delta(i) = \frac{X(i)_{predicted} - X(i)_{simulated}}{X(i)_{simulated}} \cdot 100\%$$

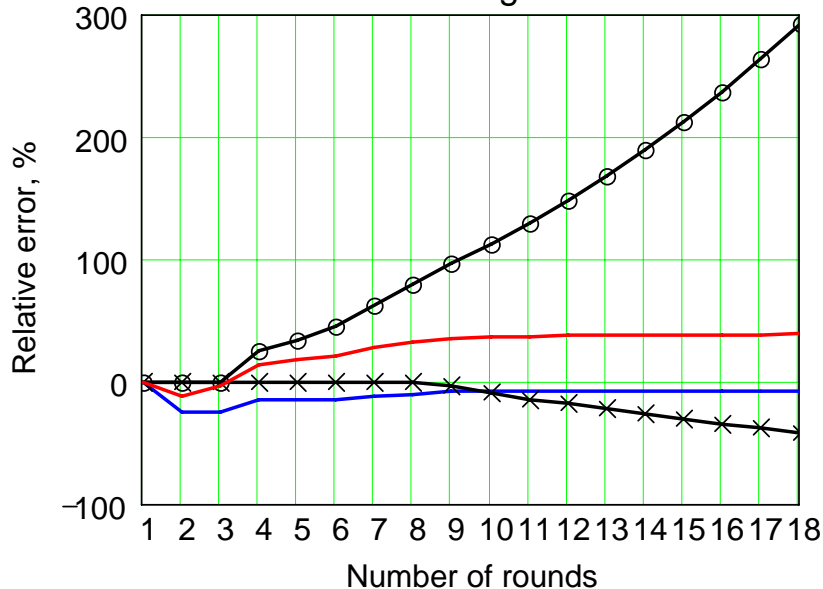
- ⌘ For $RTT > 2 \cdot T_{delACK}$, the developed model has **the relative error smaller than 10%** over a wide range of rounds and for different values of the initial window

Proposed Model (cont'd)

⌘ cwnd increase patterns, $RTT \leq T_{delACK}$

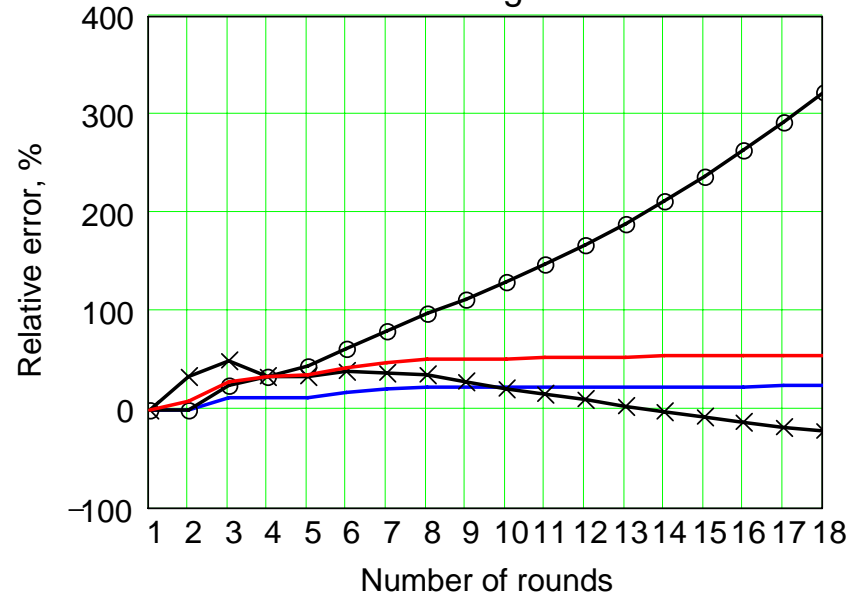
☒ The model proposed by Cardwell et al. outperforms all other models in predicting cwnd increase pattern for different values of the initial window

IW = 1 segment



— Cardwell et al
××× Sikdar et al
○-○ Zheng et al
— Proposed model

IW = 2 segments

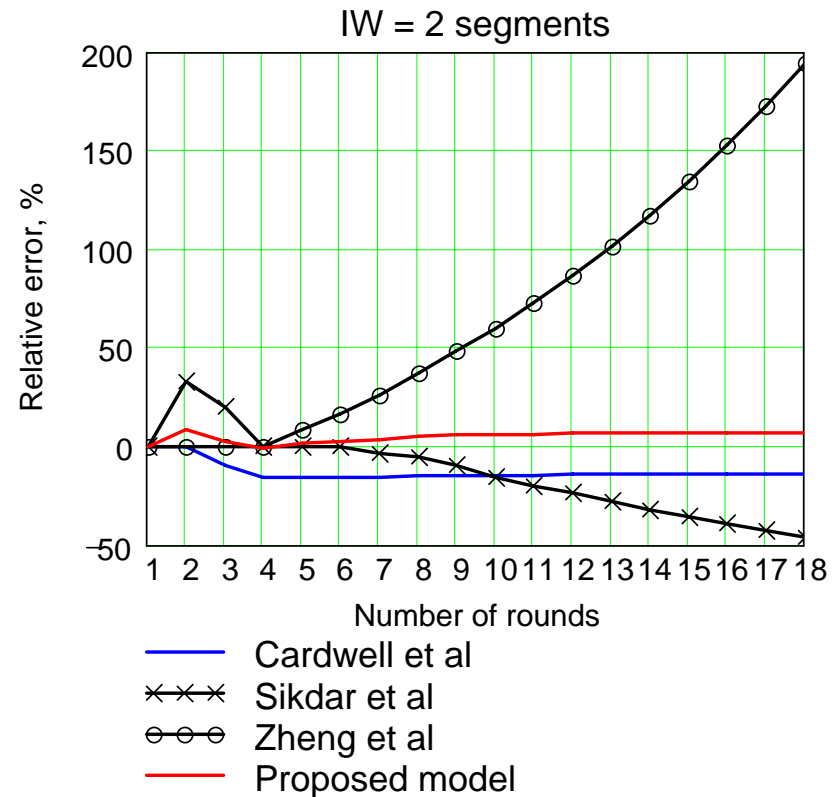
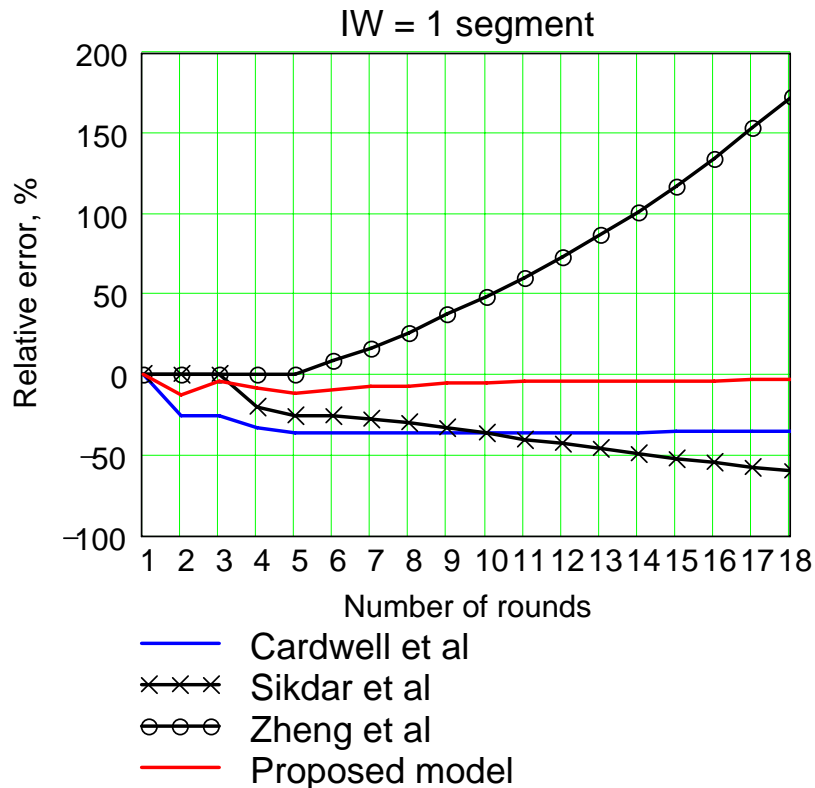


— Cardwell et al
××× Sikdar et al
○-○ Zheng et al
— Proposed model

Proposed Model (cont'd)

⌘ cwnd increase patterns, $2 \cdot T_{delACK} < RTT \leq 3 \cdot T_{delACK}$

☒ The proposed model outperforms all other models in predicting cwnd increase pattern

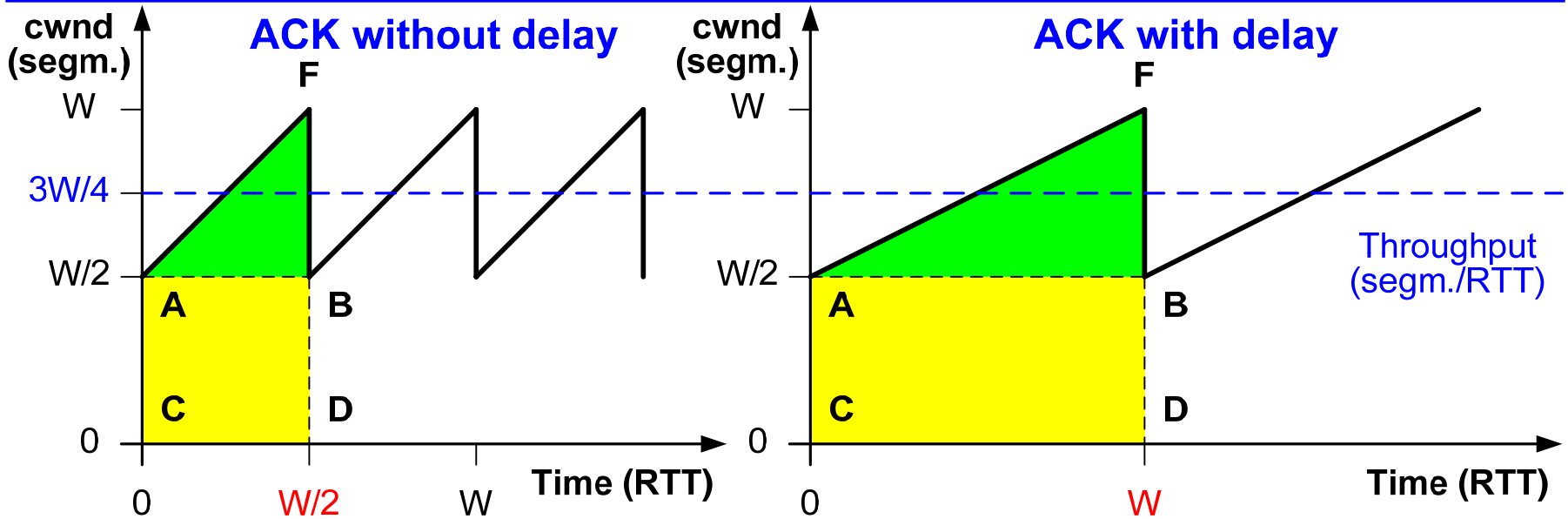


Modeling Steady State Throughput

⌘ Related work

- ⌘ M. Mathis, J. Semke, J. Mahdavi, and T. Ott. **"The macroscopic behavior of the TCP congestion avoidance algorithm"**. Computer Communication Review, volume 27, issue 3, July 1997, pp. 67-82
- ⌘ J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. **"Modeling TCP Reno performance: a simple model and its empirical validation"**. IEEE/ACM Transactions on Networking, volume 8, issue 2, April 2000, pp. 133-145
 - ☒ *Zesheng Chen, Tian Bu, Mostafa Ammar, and Don Towsley. "Comments on "Modeling TCP Reno performance: a simple model and its empirical validation"". IEEE/ACM Transactions on Networking, volume 14, issue 2, April 2006, pp. 451-453*
- ⌘ These papers provide analytical models of steady state throughput for a long-lived TCP Reno connection

Model by M. Mathis et al.



$$\begin{cases} Y_1 = S_{ABCD} + S_{ABF} = \left(\frac{W}{2}\right)^2 + \frac{1}{2}\left(\frac{W}{2}\right)^2 = \frac{3W^2}{8} \\ Y_1 = \sum_{k=1}^{\infty} (1-p_1)^{k-1} p_1 k = \frac{1}{p_1} \end{cases}$$

$$W = \sqrt{\frac{8}{3p_1}}$$

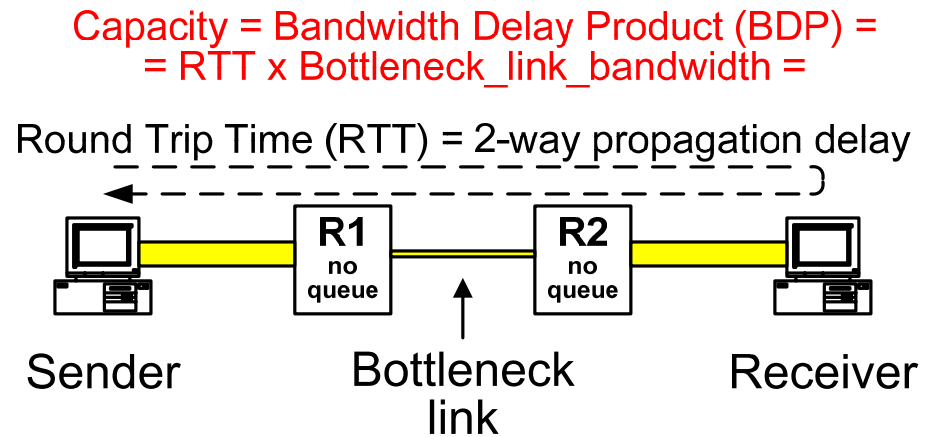
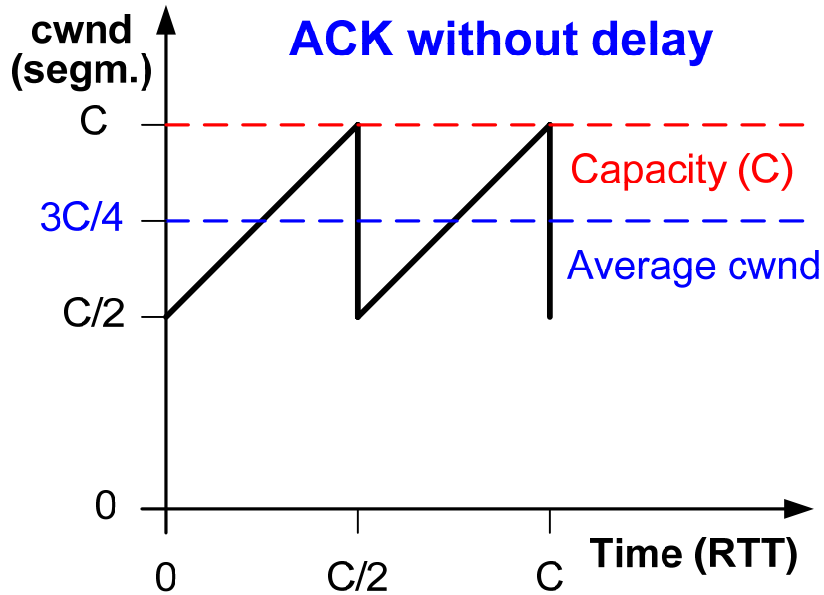
$$B_1 = \frac{Y_1}{A_1} = \frac{3}{4}W = \sqrt{\frac{3}{2p_1}}$$

$$\begin{cases} Y_2 = S_{ABCD} + S_{ABF} = \frac{W}{2}W + \frac{1}{2}\frac{W}{2}W = \frac{3W^2}{4} \\ Y_2 = \sum_{k=1}^{\infty} (1-p_2)^{k-1} p_2 k = \frac{1}{p_2} \end{cases}$$

$$W = \sqrt{\frac{4}{3p_2}} \Rightarrow \boxed{p_1 = 2p_2}$$

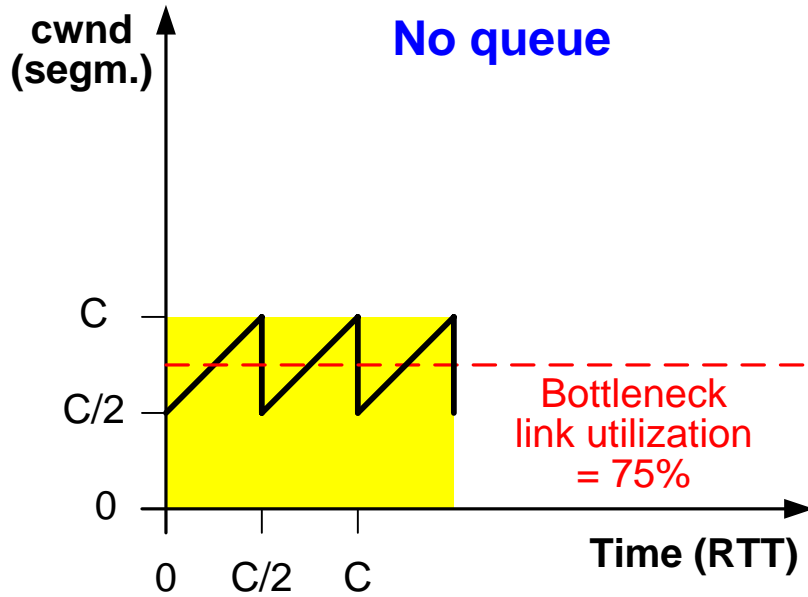
$$B_2 = \frac{Y_2}{A_2} = \frac{3}{4}W = \sqrt{\frac{3}{4p_2}} \Rightarrow \boxed{B_1(p_1) = B_2(p_2)}$$

Model by M. Mathis et al. (cont'd)

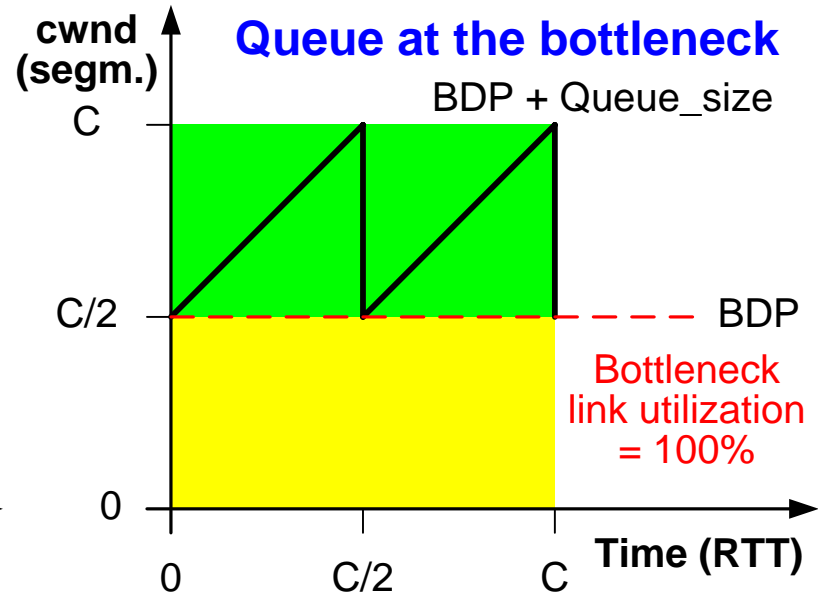


$$\text{TCP throughput} = 0.75 \times (\text{Capacity} / \text{RTT})$$

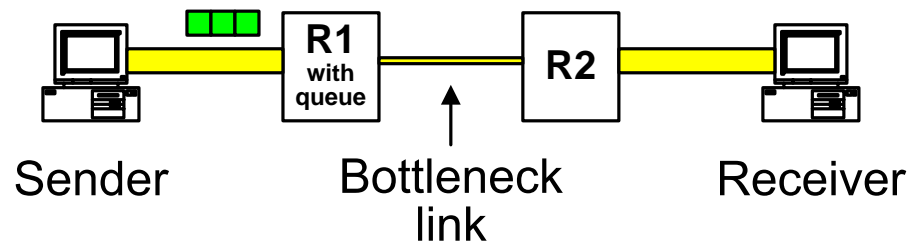
Model by M. Mathis et al. (cont'd)



Capacity = BDP



Capacity = Queue_size + BDP

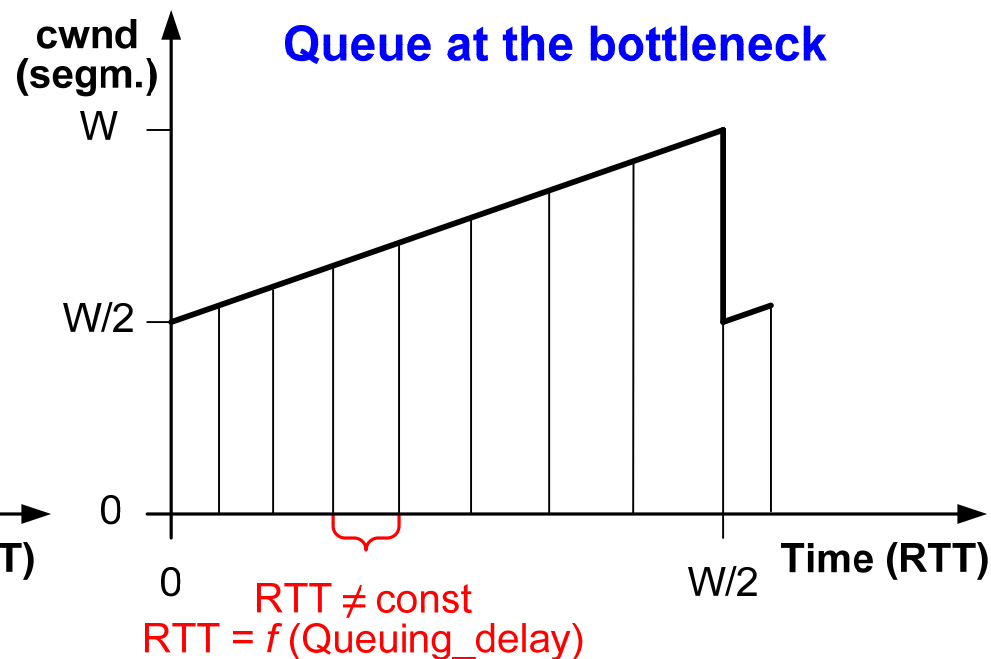
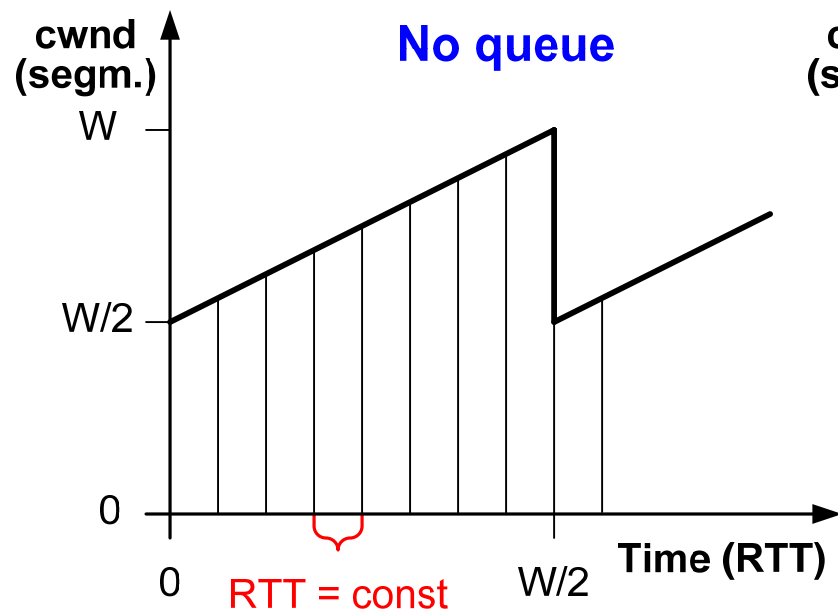


$$RTT_{\text{observed}} = \text{Queuing_delay} + 2\text{-way propagation delay}$$

Model by M. Mathis et al. (cont'd)

⌘ Limitations of the model

- ☒ No loss detection via RTO expiration
- ☒ No Karn's algorithm
- ☒ No receiver window
- ☒ No slow start algorithm
- ☒ No fast recovery algorithm



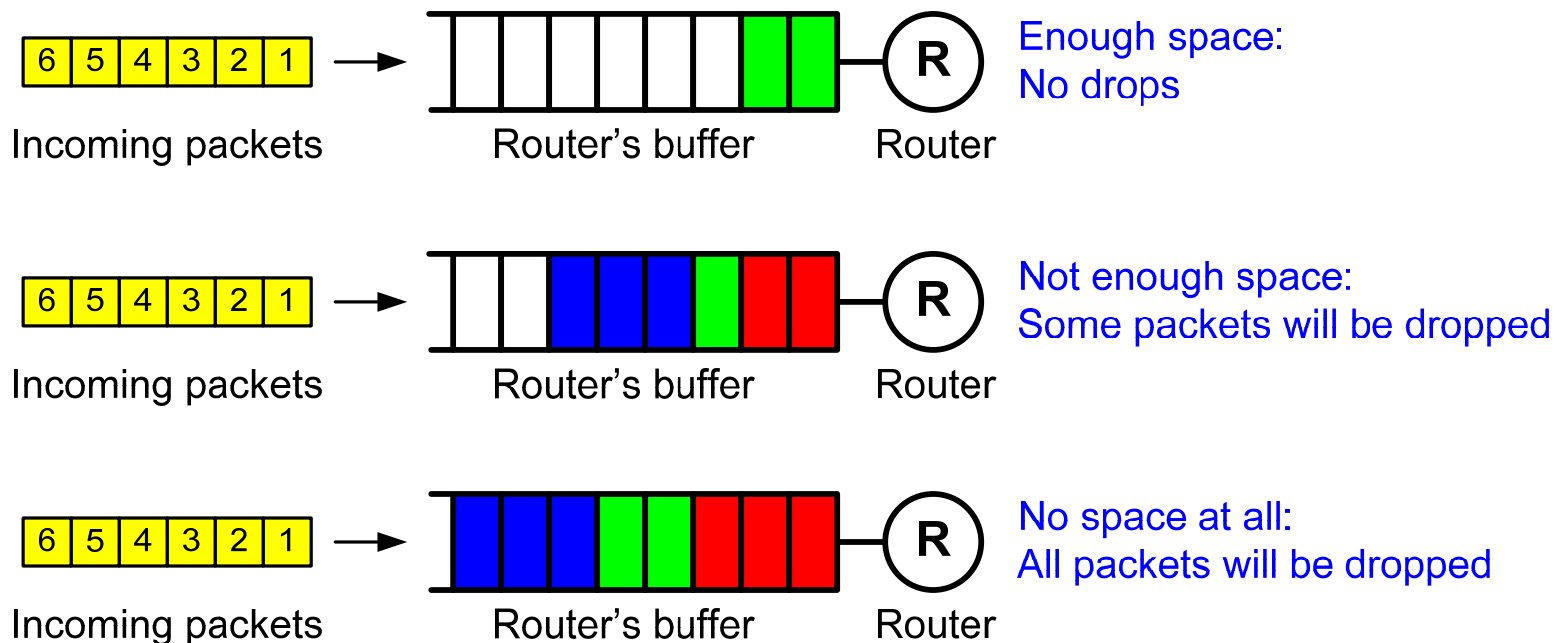
Model by J. Padhye et al.

⌘ Assumptions

- ☒ The sender sends full-sized segments whenever the congestion window allows
- ☒ The time required to send all the segments in a window is smaller than RTT
- ☒ The receiver window (rwnd) is assumed to be always constant
- ☒ Segment loss happens only in the direction from the sender to the receiver
 - ☒ This assumption is acceptable because low levels of ACK loss have only a small effect with large windows, and network paths are often much more congested in the forward direction of data flow than the direction of ACK flow (reverse direction)
- ☒ Probability of segment loss and RTT are independent of the window size
 - ☒ This can only be true when the flow is not fully utilizing the path bandwidth
 - ☒ This assumption is justified for high-speed, large BDP links with multiple flows
- ☒ **Bursty loss model:** if a segment is lost, all the remaining segments in that window are also lost
 - ☒ This model is a simplified representation of packet loss process in routers with FIFO Drop-Tail queue management

Model by J. Padhye et al. (cont'd)

⌘ Bursty loss model



Model by J. Padhye et al. (cont'd)

⌘ Model building

⌘ Three steps:

⌘ Steady state throughput is not limited by the receive window
($cwnd < rwnd$)

⊞ **Step 1:** All loss indications are exclusively “triple-duplicate” (TD) ACKs

⊞ **Step 2:** Loss indications are triple-duplicate ACKs and time-outs (TO)

⌘ **Step 3:** The impact of window limitation
($cwnd \geq rwnd$)

⌘ Terminology

⌘ **Round:** starts when the sender begins the transmission of a window of segments and ends when the sender receives an ACK for one or more of these segments

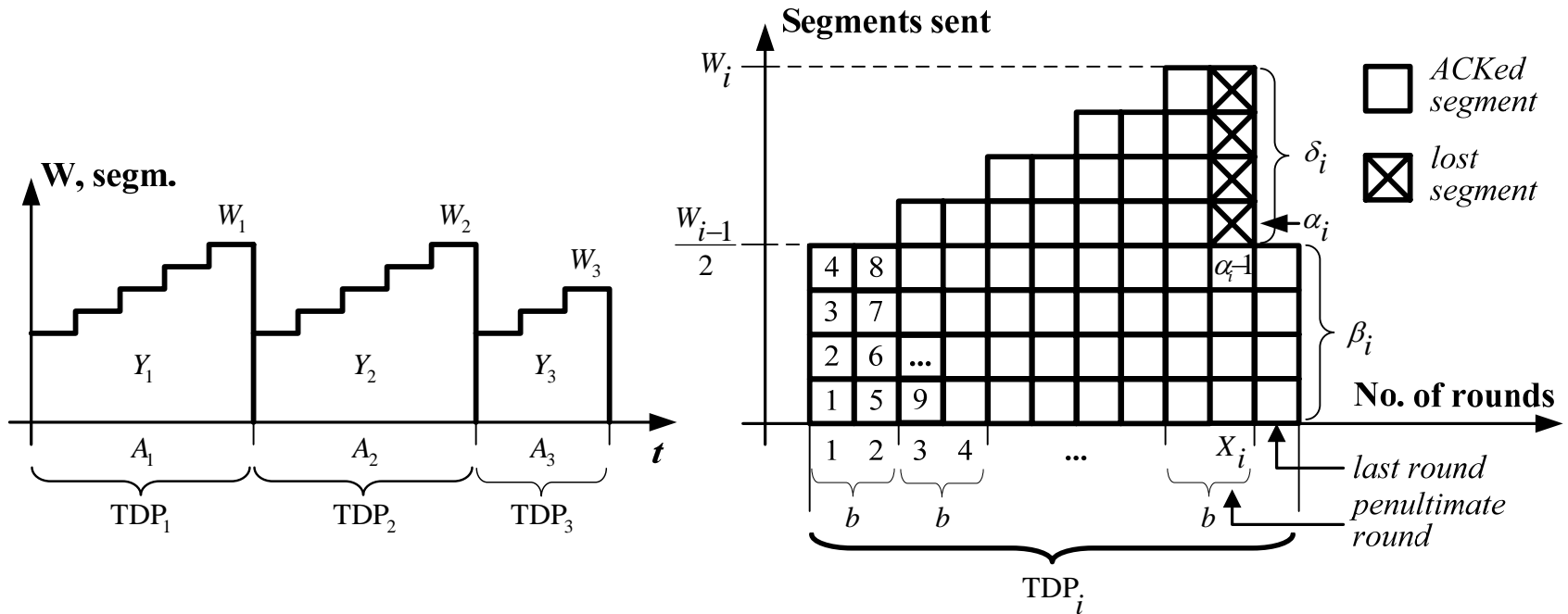
⌘ **TCP behavior as a cyclic process**

⊞ **Triple Duplicate Period (TDP):** is a period between two loss indications, except the interval between two consecutive timeouts

⊞ TDP consists only of the congestion avoidance phase

Model by J. Padhye et al. (cont'd)

⌘ Step 1: All loss indications are exclusively triple-duplicate ACKs



For any given time $t > 0$:

N_t - the number of segments transmitted in the interval $[0, t]$
(regardless of their eventual fate);

$B_t = N_t/t$ - the throughput on that interval;

$$B_t = \lim_{t \rightarrow \infty} B_t = \lim_{t \rightarrow \infty} \frac{N_t}{t}$$

For the i -th cycle (TDP_i):

Y_i - the number of segments sent in the cycle;

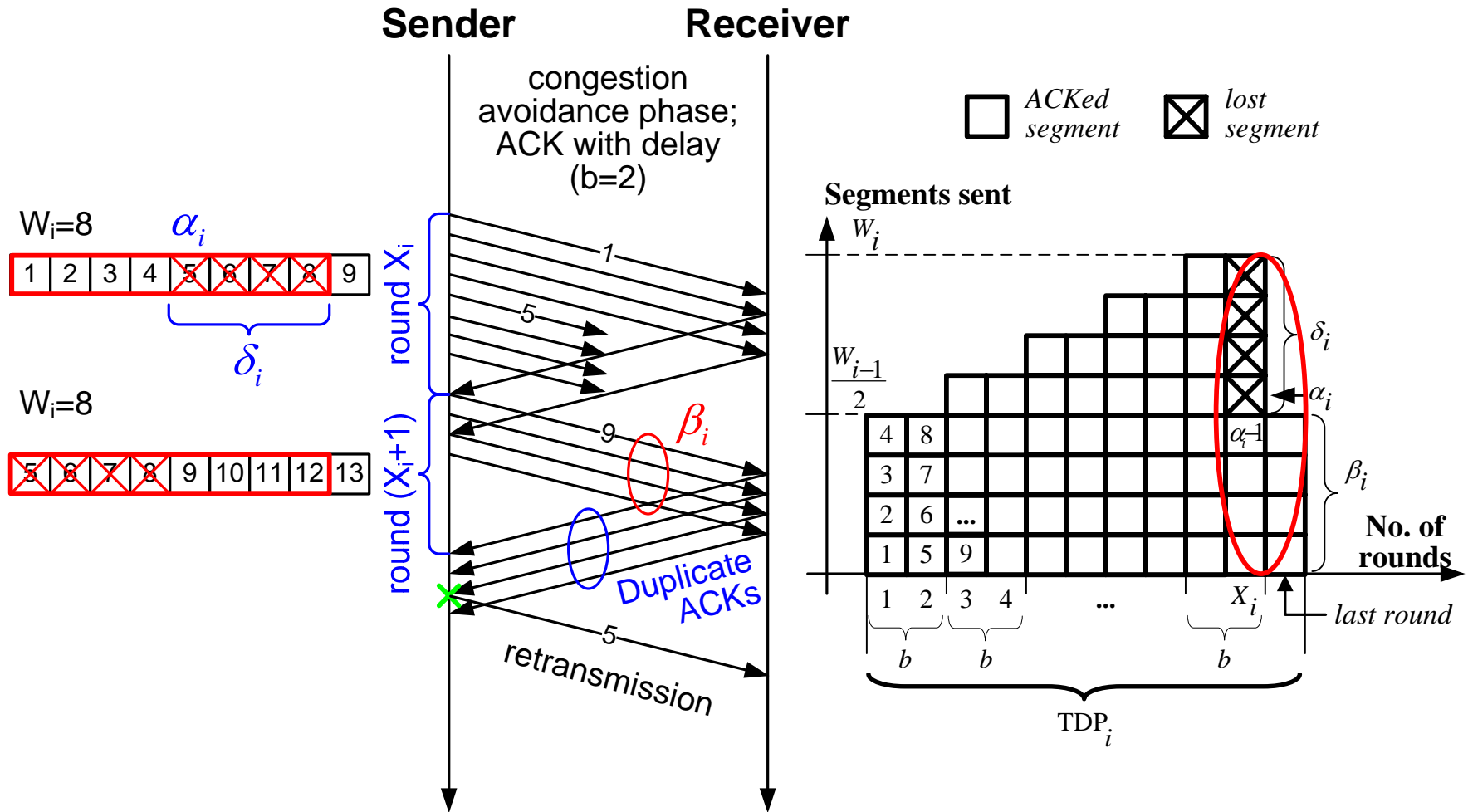
A_i - the duration of the cycle;

$\{W_i\}_i$ - a regenerative process with rewards $\{Y_i\}_i$;

$$B = \frac{E[Y]}{E[A]}$$

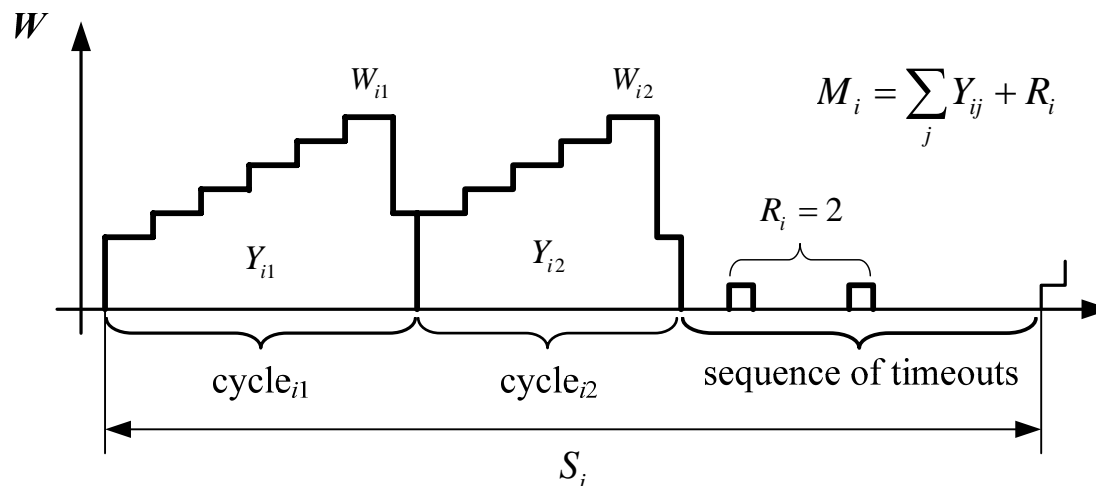
Model by J. Padhye et al. (cont'd)

⌘ End of the i -th cycle



Model by J. Padhye et al. (cont'd)

⌘ Step 2: Loss indications are triple-duplicate ACKs and time-outs



M_i - the total number of segments transmitted during the i -th supercycle;

S_i - the total duration of the i -th supercycle;

$E[Y]$ - the expected number of segments sent in a cycle;

$E[A]$ - the expected duration of a cycle;

Q - the probability that a loss indication is a TO;

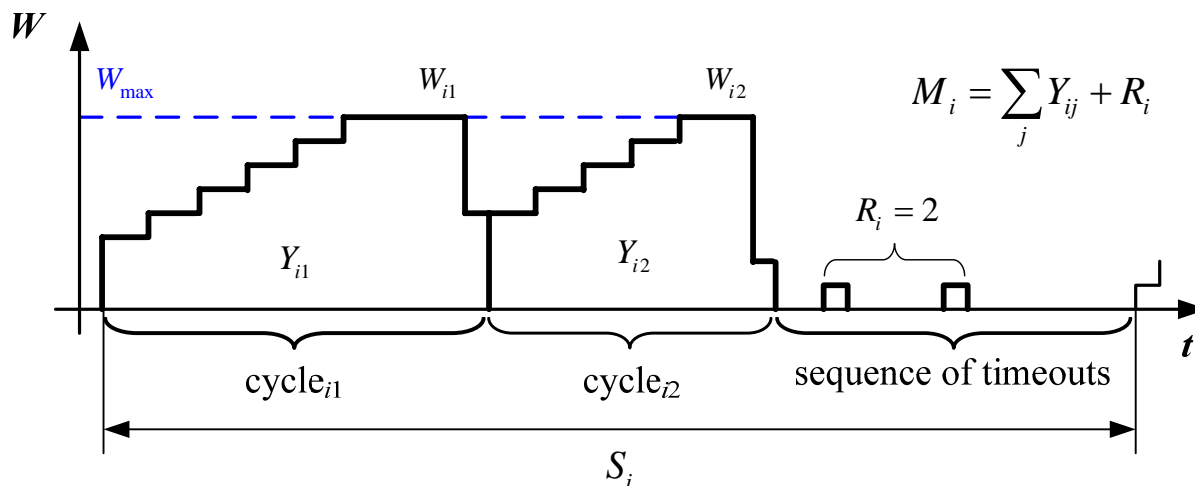
$E[R]$ - the expected number of segments sent during the timeout sequence;

$E[Z^{TO}]$ - the expected duration of the timeout sequence;

$$B = \frac{E[M]}{E[S]} = \frac{E[Y] + Q \cdot E[R]}{E[A] + Q \cdot E[Z^{TO}]}$$

Model by J. Padhye et al. (cont'd)

⌘ Step 3: The impact of window limitation



W_{\max} - the receive window size (expressed in segments);

$E[W]$ - the unconstrained window size;

$$B = \begin{cases} B(E[W]), & \text{if } E[W] < W_{\max} \\ B(W_{\max}), & \text{otherwise} \end{cases}$$

$$B = \min \left(\frac{W_{\max}}{RTT}, \frac{1}{\frac{RTT}{\sqrt{3}} \sqrt{2bp} + RTO \cdot \min \left(1, 3\sqrt{\frac{3bp}{8}} \right) p(1 + 32p^2)} \right)$$

Model by J. Padhye et al. (cont'd)

⌘ Equation-based congestion control

⊞ **RFC 3448 “TCP friendly rate control (TFRC): protocol specification”**

⌘ TFRC is designed to be reasonably fair when competing for bandwidth with TCP flows

⊞ A flow is “reasonably fair” if its sending rate is generally within a factor of two of the sending rate of a TCP flow under the same conditions

⌘ In order to compete fairly with TCP, TFRC uses the TCP throughput equation, which roughly describes TCP’s sending rate as a function of the loss event rate, round-trip time, and packet size

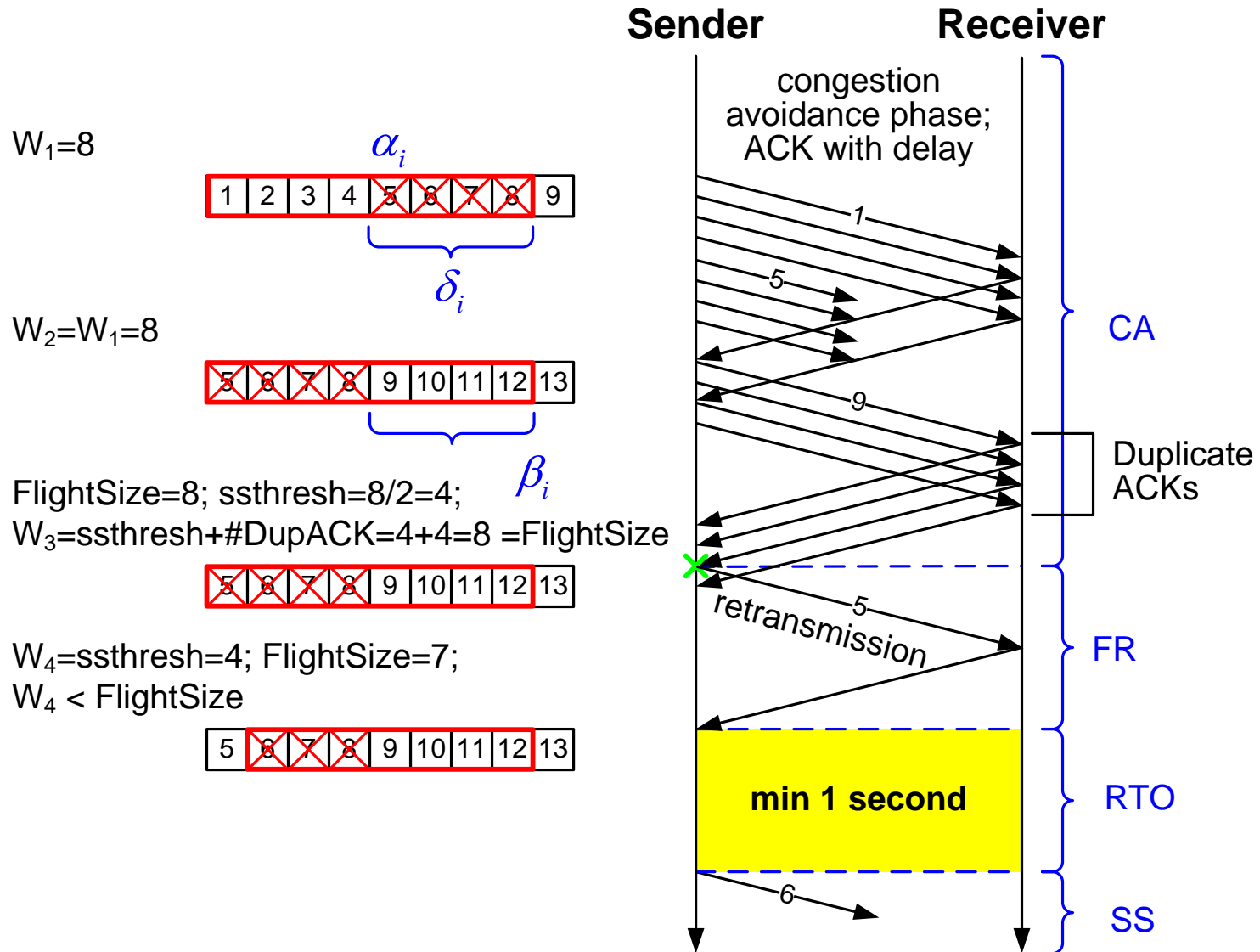
⊞ A loss event corresponds to one or more lost or marked packets from a window of data

⌘ The throughput equation is currently recommend for TFRC is a slightly simplified version of the throughput equation for TCP Reno from the model by J. Padhye et al.

⊞ Any realistic equation giving TCP throughput as a function of loss event rate and RTT should be suitable for use in TFRC

PFTK-Model Revised

⌘ TCP Reno performance in the presence of correlated losses



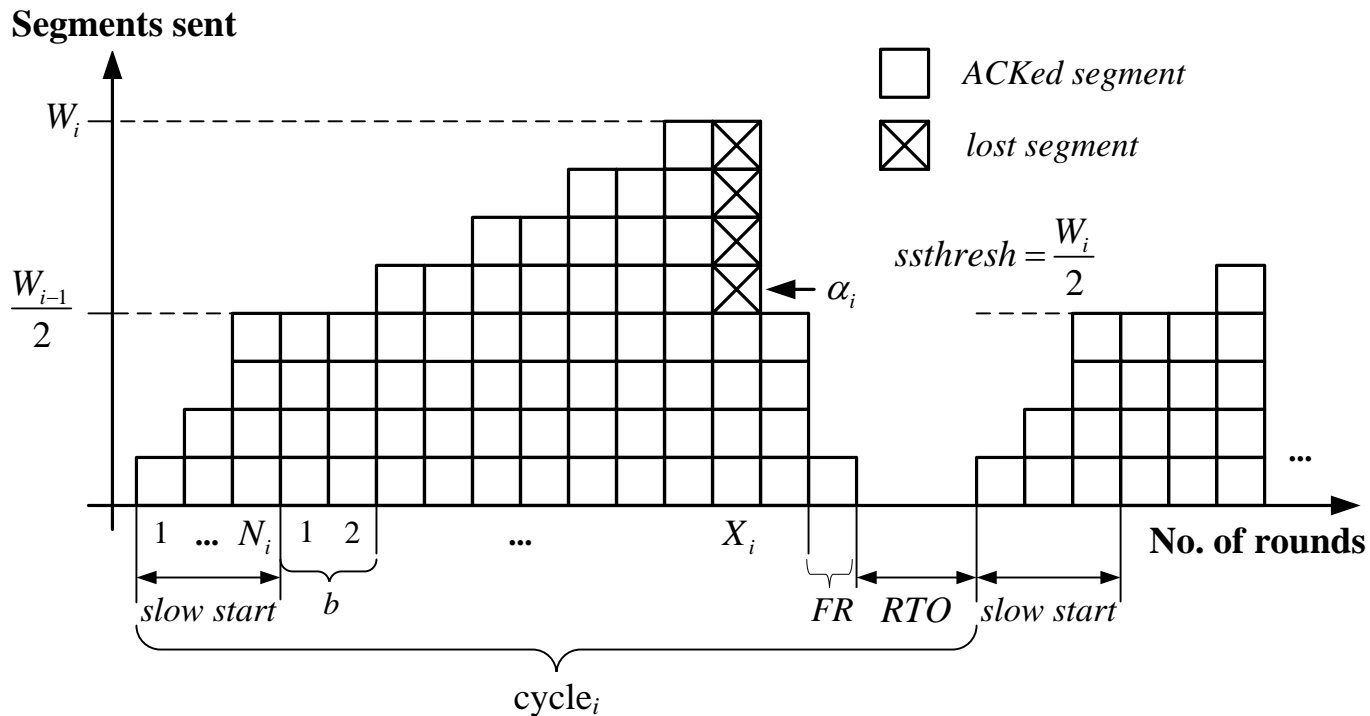
PFTK-Model Revised (cont'd)

- ⌘ The PFTK-model assumes that in case of congestion along the path about half of the transmitted window of segments will be lost
 - ☒ At the same time, the PFTK-model does not capture the fast recovery algorithm since it is assumed that the impact of this omission is quite small
 - ☒ It is also assumed that the time spent in slow start is negligible
- ⌘ However, it is known that TCP Reno has performance problems when multiple segments are dropped from one window of segments and that these problems result from the need to wait for the RTO expiration before reinitiating data flow
- ⌘ Thus, in the presence of correlated losses and when the first loss in a cycle is detected via a TD loss indication, the following sequence of steps is expected:
 - ☒ Initialization of the fast retransmit and fast recovery algorithms, retransmission of the first lost segment
 - ☒ Waiting for the RTO expiration, which was set after the successful retransmission of the first lost segment (see RFC 2988 for details)
 - ☒ Initialization of the slow start algorithm

PFTK-Model Revised (cont'd)

⌘ New definition of a cycle:

- ⊠ A cycle consists of the slow start phase, congestion avoidance phase, fast retransmit/fast recovery phase, and one timeout



PFTK-Model Revised (cont'd)

⌘ The revised model of TCP Reno throughput in the presence of correlated losses:

$$B = \begin{cases} \frac{\frac{1}{p} + E[W] + \hat{Q}(E[W]) \frac{p}{1-p}}{\overline{RTT} \left(\max(\log_2 E[W], 2) + b \left(\frac{E[W]}{2} + 1 \right) + 2 \right) + \overline{RTO} + \hat{Q}(E[W]) \left(\frac{\overline{RTO} f(p)}{1-p} - \overline{RTT} \right)}, & \text{when } W_{\max} > E[W]; \\ \frac{\frac{1}{p} + W_{\max} + \hat{Q}(W_{\max}) \frac{p}{1-p}}{\overline{RTT} \left(\max(\log_2 W_{\max}, 2) + \frac{bW_{\max}}{8} + \frac{4 + bpW_{\max}}{4pW_{\max}} + \frac{3}{2} \right) + \overline{RTO} + \hat{Q}(W_{\max}) \left(\frac{\overline{RTO} f(p)}{1-p} - \overline{RTT} \right)}, & \text{when } W_{\max} \leq E[W], \end{cases}$$

where: $E[W] = -\left(\frac{2+3b}{3b}\right) + \sqrt{\frac{8}{3bp} + \left(\frac{2+3b}{3b}\right)^2};$

$$f(p) = 2p + 2p^2 + 4p^3 + 8p^4 + 16p^5 + 32p^6;$$

$$\hat{Q}(w) = \min \left(1, \frac{\left((1 - (1-p)^3) \cdot (1 + (1-p)^3 \cdot (1 - (1-p)^{w-3})) \right)}{1 - (1-p)^w} \right).$$

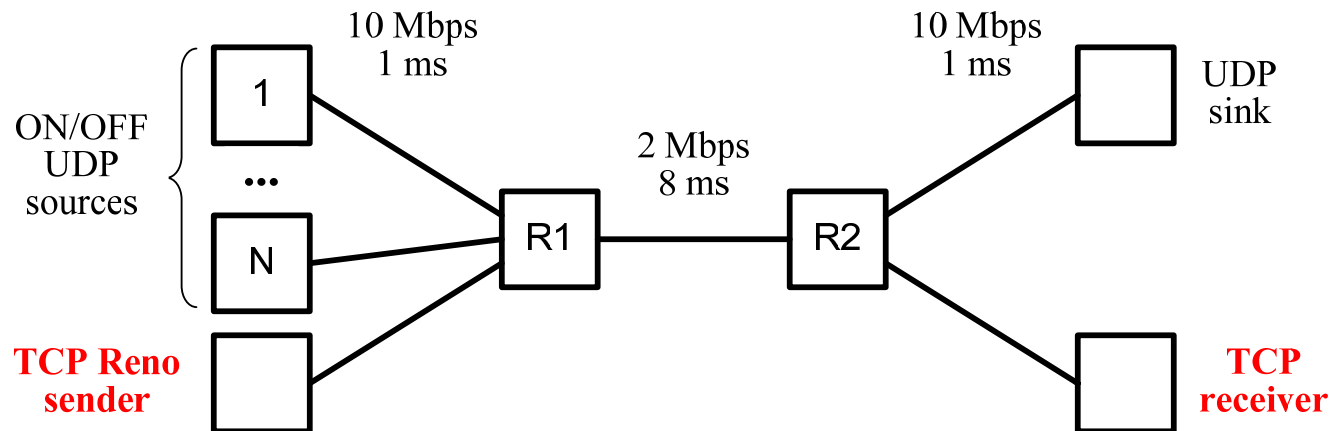
PFTK-Model Revised (cont'd)

Algorithm	Model by M. Mathis et al.	Model by J. Padhye et al.	The revised PFTK-model
Loss detection via three duplicate ACKs	Yes	Yes	Yes
Loss detection via RTO expiration	---	Yes	Yes
Karn's algorithm ("exponential backoff")	---	Yes	Yes
Sliding window	Yes	Yes	Yes
Receive window	---	Yes	Yes
Slow start	---	---	Yes
Congestion avoidance	Yes	Yes	Yes
Fast retransmit	Yes	Yes	Yes
Fast recovery	---	---	Yes

PFTK-Model Revised (cont'd)

⌘ Simulation setup

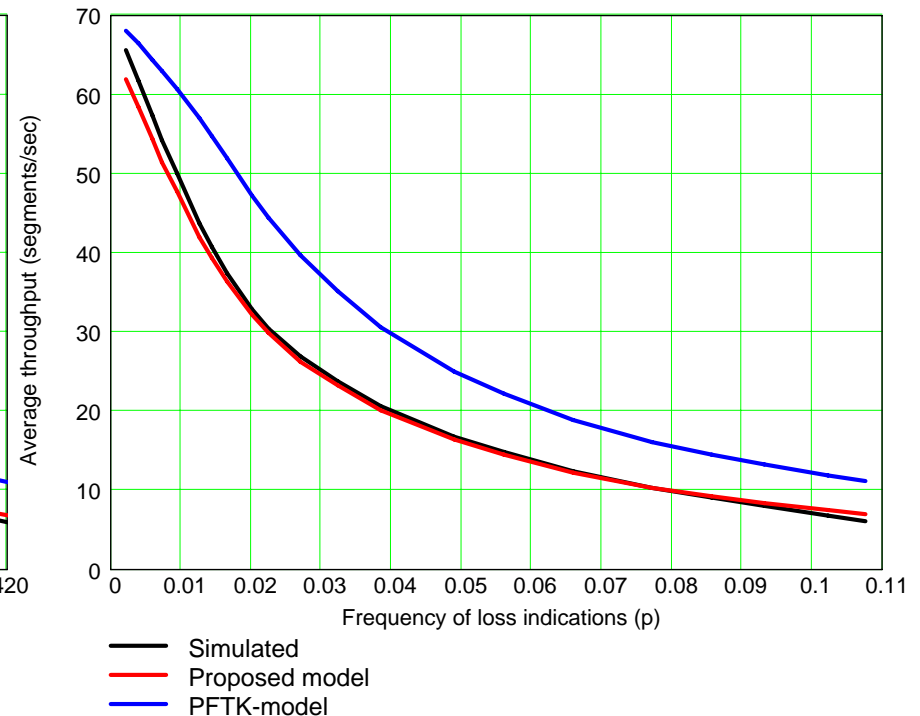
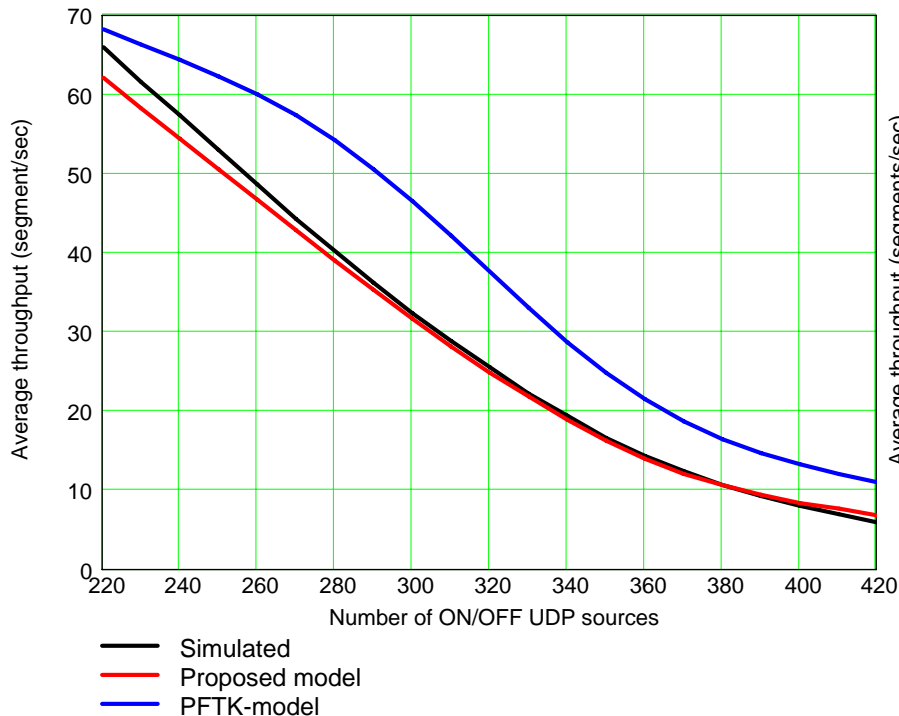
- ☒ network simulator ns-2
- ☒ TCP maximum segment size = 1460 bytes
- ☒ $W_{\max} = 10$ segments
- ☒ Number of UDP sources: from 220 to 420
 - ☒ shape parameter of Pareto distribution = 1.2
 - ☒ mean ON time = 1 s, rate = 12 kbit/s
 - ☒ mean OFF time = 2 s



PFTK-Model Revised (cont'd)

⌘ Summary

- ⌘ Fast retransmit and fast recovery algorithms should to be taken into account while modeling TCP Reno throughput in the presence of correlated losses
- ⌘ The revised model has the average error smaller than 5% over a wide range of loss rates, while the PFTK-model significantly overestimates TCP Reno throughput in the middle-to-high loss rate range

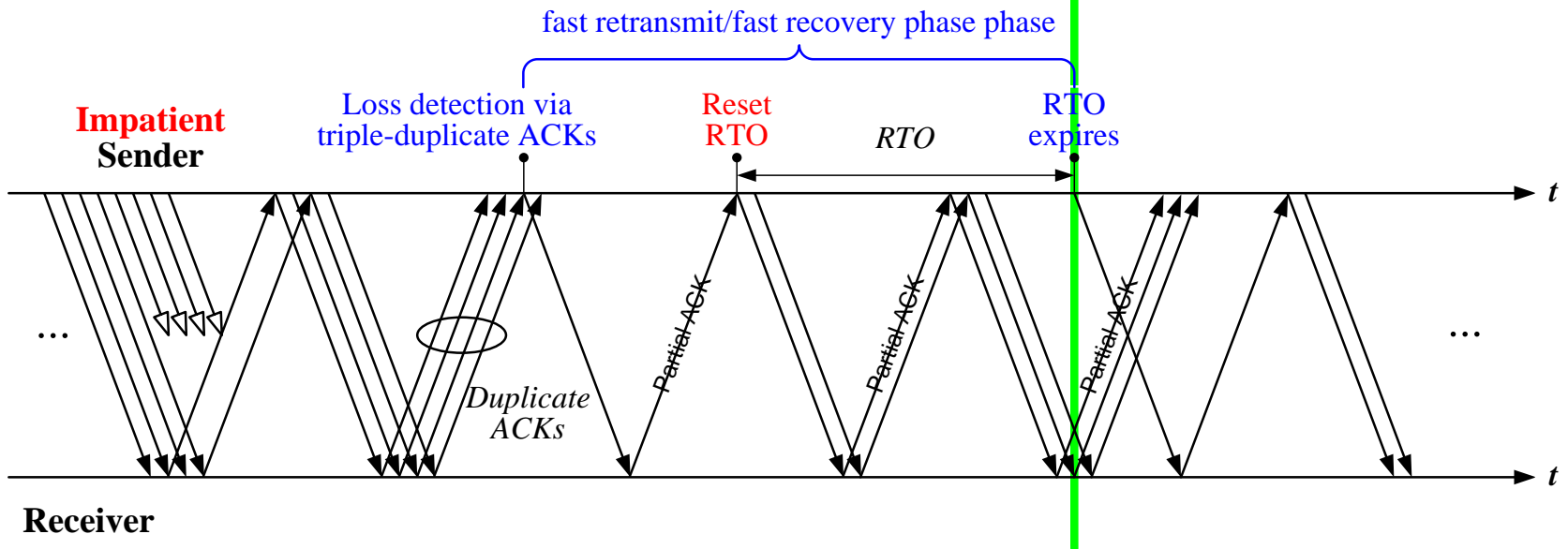
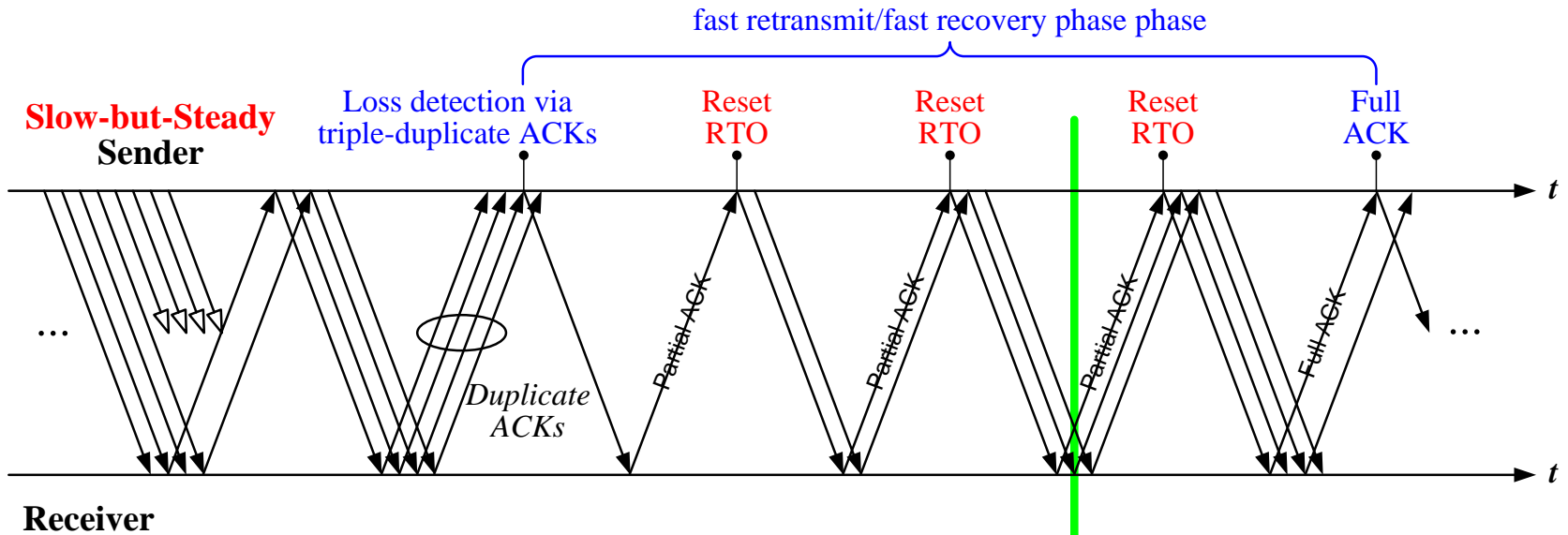


Modeling TCP NewReno

- ⌘ Related work
- ⌘ **RFC 3782 “The NewReno modification to TCP’s fast recovery algorithm”**
- ⌘ A. Medina, M. Allman, and S. Floyd. **“Measuring the evolution of transport protocol in the internet”**. ACM SIGCOMM Computer Communication Review, volume 35, issue 2, April 2005, pp. 37-52
- ⌘ N. Parvez, A. Mahanti, and C. Williamson. **“TCP NewReno: Slow-but-Steady or Impatient?”**. In Proc. of IEEE ICC 2006, June 2006

TCP stack	May 2001	February 2004
TCP NewReno	1571 (42.1%)	21266 (76.2%)
TCP Reno	667 (17.9%)	3925 (14.1%)
TCP Tahoe	201 (5.4%)	983 (3.5%)
Other	1289 (34.6%)	1378 (4.9%)
Uncategorized	no data	362 (1.3%)
Classified servers	3728	27914

Modeling TCP NewReno (cont'd)

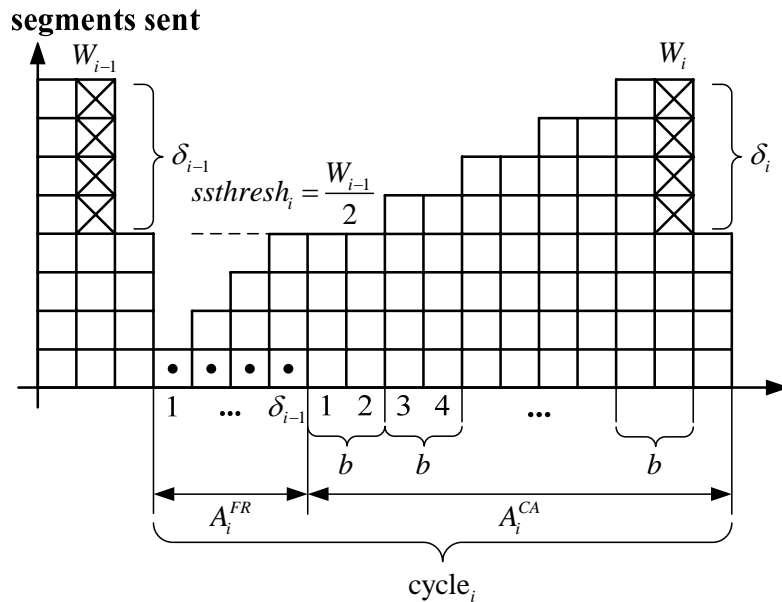


Proposed Model

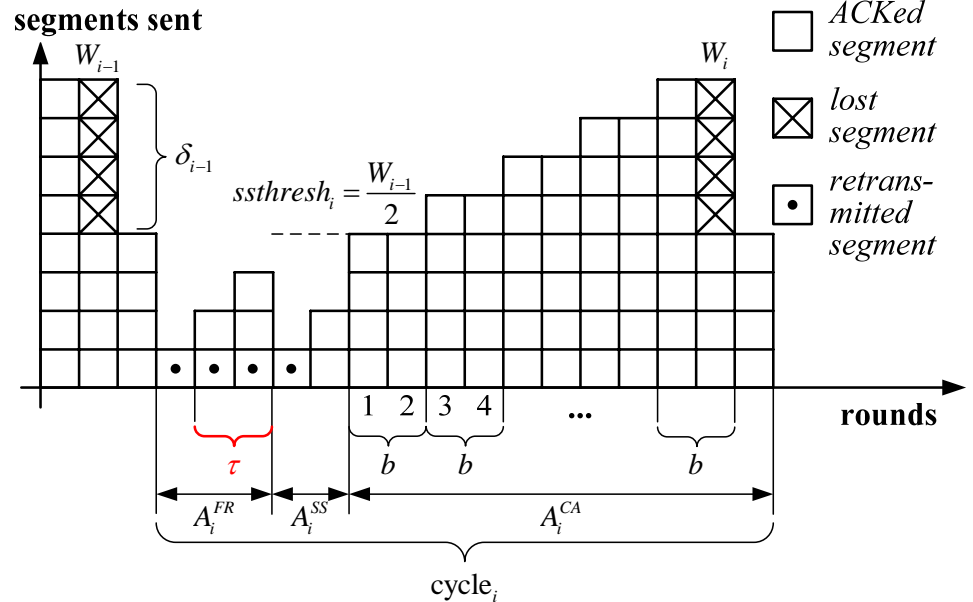
- ⌘ RFC 3782 specifies two variants of TCP NewReno implementation:
 - ☑ Slow-but-Steady
 - ☑ Impatient
- ⌘ The only difference between them lies in the retransmission timer resetting scheme in response to partial ACKs
 - ☑ In the Slow-but-Steady variant, the sender resets the retransmission timer after each partial ACK
 - ☑ In the Impatient variant, the sender performs resetting only after the first partial ACK
- ⌘ Depending on the given operational conditions (number of lost segments, delay variation, etc.) either one or the other may provide better performance
- ⌘ **The objective is to evaluate TCP NewReno variants and define the most preferable one**
 - ☑ Since the difference between them only appears when a large enough number of segments in a window are lost, we focus our analysis on case of bursty losses inherent to a Drop-Tail environment

Proposed Model (cont'd)

Slow-but-Steady



Impatient



Slow-but-Steady: $B = f(\bar{\delta}, p, b, \overline{RTT}, W_{\max})$

Impatient: $B = f(\tau, \bar{\delta}, p, b, \overline{RTT}, W_{\max})$

$$B^{IMP} = \begin{cases} B^{SBS} = \frac{E[Y]}{\overline{RTT} (E[A^{FR}] + E[Y^{CA}])}, & \text{if } \bar{\delta} < \tau + 1; \\ \frac{E[Y]}{\overline{RTT} (E[A^{FR}] + E[A^{SS}] + E[Y^{CA}])}, & \text{otherwise.} \end{cases}$$

$\bar{\delta}$ - average loss burst length

p - loss event rate

b - receiver acknowledges every b -th segm.

W_{\max} - maximum receiver window

$$\tau = \frac{\overline{RTO}}{\overline{RTT}}$$

Proposed Model (cont'd)

⌘ Numerical analysis

- ⌘ All calculated values should be real and positive by definition
- ⌘ To quantify the difference between the steady state throughputs of the Slow-but-Steady and Impatient variants we used

$$\text{Normalized difference} = \frac{(B^{IMP} - B^{SBS})}{B^{SBS}} 100\%$$

⌘ Used parameters

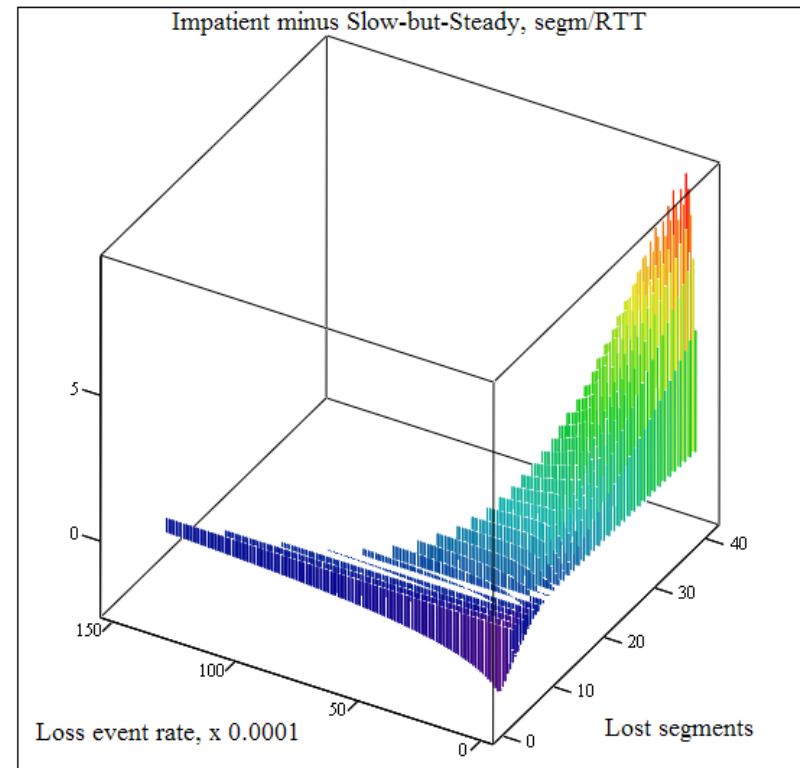
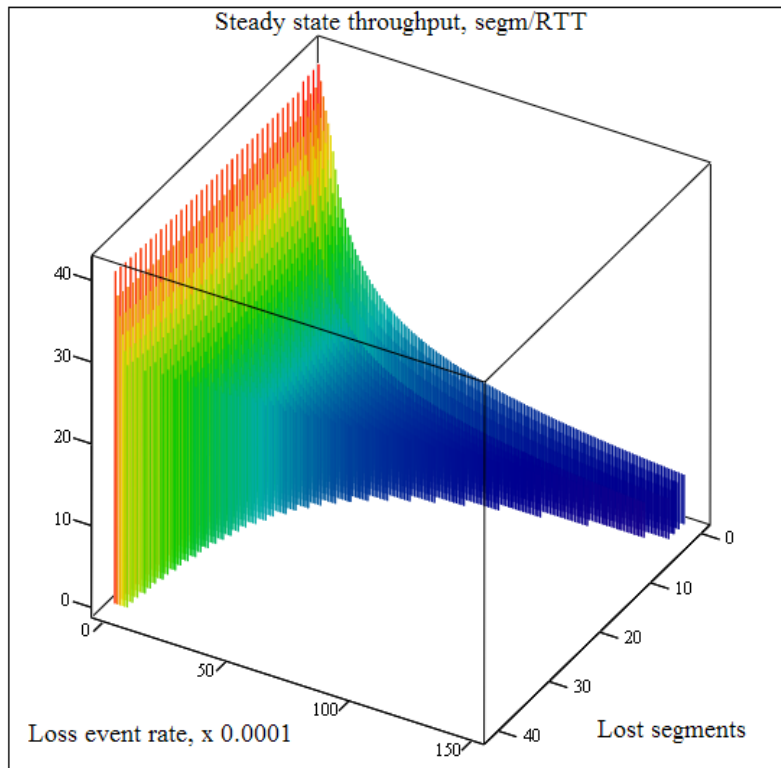
- ⌘ $b = 1$ or 2 (without/with delayed acknowledgements)
- ⌘ Receiver buffer size = 64 KB (commonly, TCP adjusts W_{\max} to even increments of MSS negotiated during connection establishment)
- ⌘ MSS = 1460 bytes (then $W_{\max} = 44$ segments)
- ⌘ τ varied from 2 up to 20 (recall that $\tau = RTO/RTT$)
- ⌘ p varied from 0.01 up to 1.5% (higher values of p lead to timeouts)
- ⌘ $\bar{\delta}$ varied from 1 up to 41 segments ($\bar{\delta} \leq W_{\max} - 3$)

Proposed Model (cont'd)

- ⌘ Steady state throughput of the Impatient variant and difference between throughputs of the Impatient and Slow-but-Steady variants

$$b = 2; \overline{RTO} = 4\overline{RTT}; MSS = 1460 \text{ bytes}; W_{\max} = 44 \text{ segm.}$$

- ⌘ Throughput shows a complex behavior as a function of δ , ρ , and τ

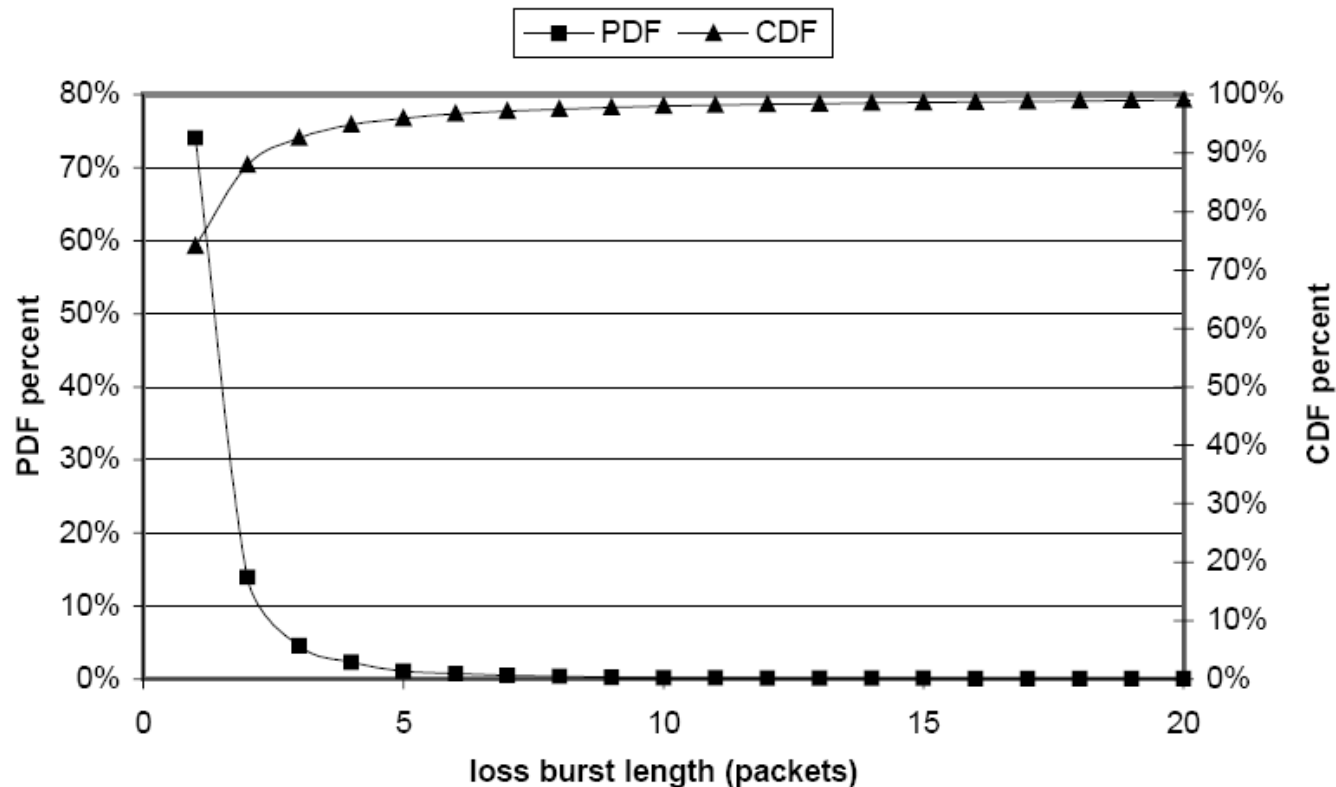


Proposed Model (cont'd)

- ⌘ As it was noted in RFC 3782, neither of the two variants is optimal:
 - ☒ When the number of lost segments is small the performance would have been better without invocation of slow start algorithm
 - ☒ Slow-but-Steady outperforms Impatient by **17.8%** (by 4.2 segm./RTT)
 - ☒ At the same time, when the number of lost segments is sufficiently large, the Impatient variant gives a faster recovery and better performance and the gain increases with the average loss burst length
 - ☒ Impatient outperforms Slow-but-Steady by **90.6%** (by 14.5 segm./RTT)

Proposed Model (cont'd)

- ⌘ D. Loguinov. **"Adaptive scalable Internet streaming"**. PhD thesis, 2002
- ⌘ 1) The majority of losses are single packet losses
- ⌘ 2) Most TCP implementations use coarse-grained retransmission timer (e.g., according to RFC 2988, RTO should be at least 1 second)
- ⌘ As a result, the Impatient variant will behave like the Slow-but-Steady one (since all lost segments can be recovered using fast recovery algorithm before RTO expiration)



Proposed Model (cont'd)

⌘ Summary

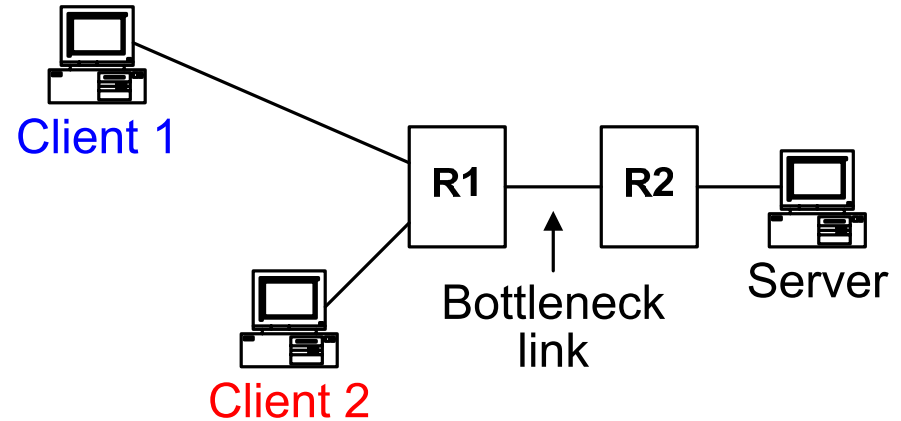
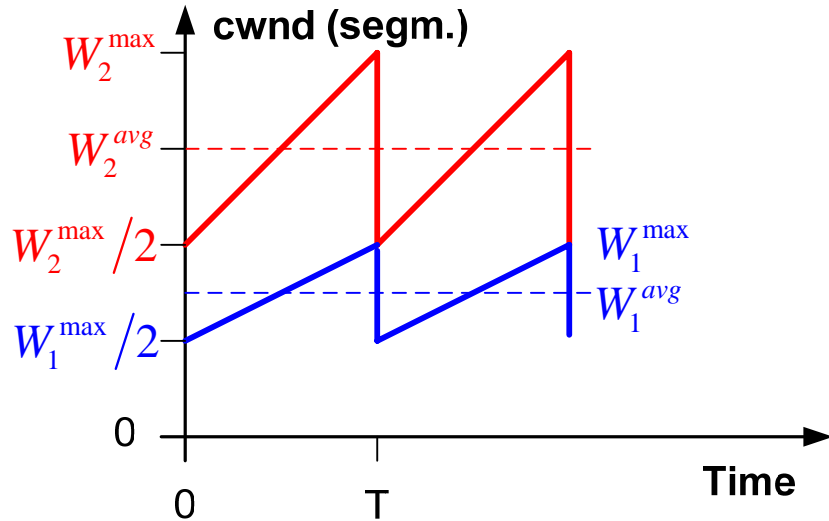
- ☒ The Impatient variant provides approximately the same steady state throughput as the Slow-but-Steady one in the worst case and significantly outperforms it in case of large windows and bursty losses
 - ☒ This could be extremely useful for networks with large bandwidth and long delay
- ☒ It is expected that under normal operational conditions there will be no difference between the Impatient and Slow-but-Steady variants since in the most cases all lost segments can be recovered in the Slow-but-Steady mode
- ☒ **Our recommendation is for the Impatient** variant as a backup mechanism for extreme scenarios with multiple packet drops

TCP Fairness

⌘ Related work

- ⌘ J. Lee, S. Bohacek, J. Hespanha, and K. Obraczka. **“A study of TCP fairness in high-speed networks”**. University of Southern California, Technical Report 05-854, 2005
- ⌘ This paper provides a simple analytical model that characterizes fairness under drop synchronization
 - ⊞ I.e., when competing TCP flows become synchronized in the sense that they suffer drops roughly at the same time
- ⌘ When several TCP flows with different RTTs share the same bottleneck link, they will not gain access to an equal share of the available bandwidth
 - ⊞ This “unfairness” is caused by TCP’s AIMD (Additive Increase Multiplicative Decrease) scheme in the congestion avoidance stage
- ⌘ The AIMD algorithm in TCP increases the congestion window by one for each RTT and decreases the window by half when a drop is detected
 - ⊞ Flows with smaller RTT increase their congestion window more rapidly and are therefore able to reach larger sending rates before congestion occurs
 - ⊞ Flows with larger RTTs will thus achieve smaller average sending rates

TCP Fairness (cont'd)



W_i - the window size of the i -th flow;

T - loss period;

$\frac{1 \text{ segm.}}{RTT_i}$ - the rate of increase of W_i ;

$$\frac{W_i^{\max}}{2} + \frac{T}{RTT_i} = W_i^{\max} \Rightarrow W_i^{\max} = \frac{2T}{RTT_i}$$

$$W_i^{\text{avg}} = \frac{1}{2} \left(\frac{W_i^{\max}}{2} + W_i^{\max} \right) = \frac{3}{4} W_i^{\max} = \frac{3}{2} \frac{T}{RTT_i}$$

$$B_i = \frac{W_i^{\text{avg}}}{RTT_i} = \frac{3}{2} \frac{T}{RTT_i^2}$$

$$\text{Fairness}_{1,2} = \frac{B_1}{B_2} = \left(\frac{RTT_2}{RTT_1} \right)^2$$

Ratio between throughputs of two flows is inversely proportional to the square of ratio of their RTTs

Network Asymmetry & TCP

⌘ Related work

⌘ **RFC 3449 “TCP performance implications of network path asymmetry”**

☒ The document describes TCP performance problems that arise because of asymmetric effects and details several mitigations to these effects, which have either been proposed or evaluated in the literature, or are currently deployed in networks

⌘ T. V. Lakshman, U. Madhow, and B. Suter. **“Window-based error recovery and flow control with a slow acknowledgement channel: a study of TCP/IP performance”**. In Proc. of IEEE INFOCOM 1997, volume 3, issue 7-12, April 1997, pp. 1199-1209

☒ The paper evaluates the performance of window-based protocols over systems in which the reverse link is considerably slower than the forward link

Network Asymmetry & TCP (cont'd)

- ⌘ **The bandwidth asymmetry has a negative impact on TCP performance** because the TCP throughput is regulated (and limited) by the flow of acknowledgements (called "ACK clock")
- ⌘ The impact of asymmetry on TCP performance has been characterized by Lakshman et al. using an index called **normalized asymmetry ratio (k)**
- ⌘ k is the ratio of the raw bandwidth (capacity) of the forward direction to the reverse direction, divided by the ratio of the packet sizes used in the two directions

$$k = \frac{\frac{\text{forward_link_bandwidth}}{\text{reverse_link_bandwidth}}}{\frac{\text{forward_packet_size}}{\text{reverse_packet_size}}}$$

- ⌘ When $k > 1$, the TCP throughput on the forward channel is restricted to a maximum of **forward_link_bandwidth / k**
- ⌘ Other factors like bi-directional traffic (e.g., downloading a Web page while sending an email) will increase k and further aggravate the asymmetry problem

Network Asymmetry & TCP (cont'd)

⌘ Example

⌘ Forward link bandwidth = 8 Mbit/s

⌘ Reverse link bandwidth = 64 kbit/s

☒ The raw bandwidth ratio is $8000/64 = 125$

⌘ Data packets = 1500 bytes

⌘ ACK packet = 40 bytes

☒ The ratio of the packet sizes is $1500/40 = 37.5$

⌘ $k = 125/37.5 \approx 3.3$

⌘ If the receiver acknowledges more frequently than one ACK per every 3 data packets, the upstream link will become saturated before the downstream link, limiting the throughput in the forward direction

⌘ Thus, the TCP throughput on the forward link is restricted to a maximum of $8/k = 2.4$ Mbit/s

